Septièmes Rencontres nationales des Jeunes Chercheurs en Intelligence Artificielle

RJCIA 2005

Dans le cadre de la plate-forme AFIA 2005

du 1er au 3 juin à Nice

Président du comité de programme
Emmanuel Guéré

Président de la plate-forme AFIA : Fabien Gandon (INRIA Sophia Antipolis)
Site Web de la plate-forme : http://www-sop.inria.fr/acacia/afia2005/

Avant-propos

Les Rencontres Jeunes Chercheurs en Intelligence Artificielle (RJCIA) s'adressent explicitement aux chercheurs en IA qui sont en début de carrière : c'est-à-dire aux doctorants ou aux titulaires d'un doctorat depuis moins d'un an. La vocation de cette manisfestation est triple. Tout d'abord, permettre aux jeunes chercheurs de se rencontrer et de présenter leurs travaux. Ensuite former ces mêmes chercheurs à la préparation d'un article ainsi qu'à la révision de cet article en tenant compte des observations émanant du comité de programme. Enfin, ces rencontres sont l'occasion d'offrir un panorama des recherches récentes dans le domaine de l'IA en France.

Ces rencontres ont vu le jour à Rennes en 1992 à l'initiative particulière de Marie-Odile Cordier (IRISA Rennes) et de Jean-Paul Krivine (DER EDF Clamart) soutenue par l'Association Française d'Intelligence Artificielle (AFIA). Depuis lors ces rencontres organisées par l'AFIA se sont tenues tous les deux ans : Marseille (1994), Nantes (1996), Toulouse (1998), Lyon (2000) et Laval (2003). Les septièmes rencontres se dérouleront cette année à Nice du 1er au 3 juin. Pour la seconde fois, elles sont organisées conjointement aux conférences CAp (Conférence d'Apprentissage) et IC (Ingénieurie des connaissances) et supportées par la plate-forme AFIA.

Le comité de programme est constitué de chercheurs n'ayant soutenu leur thèse que depuis quelques années. Aidé de quelques relecteurs, il a fourni une évaluation scientifique particulièrement rigoureuse des articles proposés à la publication. Ainsi chaque article a été relu par au moins deux relecteurs du domaine. A l'issue des délibérations, 22 articles ont été acceptés en version longue, soit un taux d'acceptation de 55%. Ce taux relativement bas n'est dû qu'au nombre élevé de soumissions. Afin de publier le maximum de contributions de qualité, 6 articles sont publiés en version courtes et seront présentés sous la forme de posters.

Je tiens à remercier sincèrement tous ceux qui ont permis l'organisation de ces rencontres, plus particulièrement Fabien Gandon (INRIA Sophia Antipolis) président de la plate-forme AFIA 2005, ainsi que tout le comité d'organisation de la plate-forme. Je tiens également à remercier tout spécialement Florence Dupin de Saint-Cyr (IRIT Toulouse) pour sa confiance et son aide quant à la constitution de ce comité de programme de qualité reflétant le paysage de l'intelligence artificielle en France.

Emmanuel Guéré
ILOG S.A.

Comité de programme :

Président : Emmanuel Guéré	(ILOG S.A. Gentilly)
Samir Aknine	(LIP6 Paris)
Leila Amgoud	(IRIT Toulouse)
Gilles Audemard	(CRIL Lens)
Yann Chevaleyre	(LAMSADE Paris)
Gilles Dequen	(LaRIA Amiens)
Laurent Garcia	(LERIA Angers)
Pascal Garcia	(IRISA Rennes)
David Langlois	(LORIA Nancy)
Benjamin Habegger	(GRAPPA Lille)
Souhila Kaci	(CRIL Lens)
Yannick Larvor	(France Telecom R&D Lannion)
Dominique Longin	(IRIT Toulouse)
Nicolas Maudet	(LAMSADE Paris)
Laurent Perrussel	(IRIT Toulouse)
Thierry Petit	(Ecole des Mines de Nantes)
Laurent Prévot	(LOA Trento)
David Renaudie	(Leibniz Grenoble)
Régis Sabbadin	(INRA Toulouse)
Nicolas Sabouret	(LIP6 Paris)
Igor Stéphan	(LERIA Angers)
Olivier Spanjaard	(LIP6 Paris)
Cyril Terrioux	(LSIS Marseille)
Dominique Van Zwynsvoorde	(Cybernetix Saclay)
Vincent Vidal	(CRIL Lens)
Bruno Zanuttini	(GREYC Caen)

aidé des rapporteurs complémentaires :

Mathieu d'Aquin	(LORIA Nancy)
Vincent Colotte	(LORIA Nancy)
Roger Martin-Clouaire	(INRA Toulouse)
Damien Pellier	(Leibniz Grenoble)
Serge Stinckwich	(GREYC Caen)
Nicolas Prcovic	(LSIS Marseille)
François Rioult	(GREYC Caen)
Yasmine Charif	(LIP6 Paris)
Amalia Todirascu-Courtier	(Université de Troyes)
Laurent Péret	(INRA Toulouse)
Marc Pauly	(IRIT Toulouse)
Wided Lejouad Chaari	(ENSI Tunisie)
Sabine Moisan	(INRIA Sophia Antipolis)

Programme des rencontres :

Mercredi 1er juin 2005

9h30-11h00 : Diagnostic et planification

11h30-13h00 : Fouille et extraction de connaissances

14h30-16h00 : Systèmes multi-agents et Apprentissage

16h30-18h00 : Systèmes multi-agents et autonomie

Jeudi 2 juin 2005

Session plénière : Article primé par l'AFIA

Session poster : Articles courts

Vendredi 3 juin 2005

9h30-11h00 : Logique propotionnelle et contraintes temporelles

11h30-13h00 : Contraintes et similarités

14h30-16h00 : Représentation des connaissances

Pilotage en ligne d'algorithmes de traitement du signal guidé par le contexte courant

François Portet[1,2], Guy Carrault[2], Marie-Odile Cordier[1], René Quiniou[1]

[1] Institut de Recherche en Informatique et Systèmes Aléatoires
INRIA / Université de Rennes 1, Campus de Beaulieu, 35042 Rennes
{francois.portet,cordier,quiniou}@irisa.fr

[2] Laboratoire de Traitement du Signal et de l'Image, Unité INSERM 642
Université de Rennes 1, Campus de Beaulieu, 35042 Rennes
guy.carrault@univ-rennes1.fr

Résumé :

Le travail présenté fait partie d'un projet dédié au développement d'un système de monitoring des arythmies cardiaques. Dans ce type d'application, la réduction du nombre d'erreurs de détection et de diagnostic médical dues au bruit de ligne est un objectif majeur. Pour atteindre cet objectif, nous proposons de *piloter* les algorithmes de traitement du signal par le *contexte courant* constitué par l'analyse du bruit de ligne et des arythmies en cours de reconnaissance. Grâce à la connaissance de ce contexte, le pilote choisit et paramètre les algorithmes pour augmenter les performances du système de monitoring. Le contexte permet aussi de détecter les cas défavorables dans lesquels certaines *tâches* de détection d'événements commettent trop d'erreurs. Le pilote choisit alors de baser la reconnaissance d'arythmies sur les événements correctement détectables. De cette façon, la caractérisation de l'arythmie s'appuie uniquement sur les informations les plus fiables. Les décisions prises par le pilote dépendent du contexte courant et reposent sur une expertise du domaine du traitement du signal et du domaine médical. Les premiers résultats sur le pilotage de la détection d'événements montrent la faisabilité et l'intérêt d'une telle approche.

Mots-clés : pilotage d'algorithmes, monitoring cardiaque, surveillance de systèmes dynamiques, reconnaissance de chroniques, diagnostic, applications de l'I.A.

1 Introduction

Les systèmes de monitoring médical, tels que *Guardian* (Larsson & Hayes-Roth, 1998) et *Calicot* (Carrault *et al.*, 2003), comprennent généralement deux parties distinctes : une partie abstraction temporelle dédiée à l'acquisition, au traitement et à l'analyse du signal, et une partie diagnostic médical qui infère un diagnostic à partir des informations transmises par l'abstraction temporelle et d'une base de connaissance.

Dans ces systèmes, l'état courant du diagnostic médical n'est pas mis à profit pour optimiser l'abstraction temporelle. Pourtant, l'analyse de la ligne et le diagnostic médical courant sont des informations utiles pour adapter les algorithmes de traitement du signal qui constituent l'abstraction temporelle (Portet *et al.*, 2005). Soulas *et al*. (Soulas *et al.*, 1998) proposent un système de monitoring cardiaque qui utilise la différence de performance de deux algorithmes de détection d'événements. Selon le type de bruit de ligne, le signal à traiter est dirigé vers l'algorithme le plus approprié. Mais ce système possède une architecture statique qui rend difficile l'intégration de nouveaux algorithmes. De plus, le diagnostic médical n'est pas pris en compte pour la sélection.

Pour modifier dynamiquement l'abstraction temporelle, nous proposons une approche de type *pilotage d'algorithmes* dont l'architecture permet d'adapter en ligne une chaîne de traitement selon le *contexte courant* de son utilisation. Le pilotage d'algorithmes s'inspire de différents travaux de la littérature.

Shekhar *et al.* (Shekhar *et al.*, 1994) introduisent le pilotage de programmes qui représente une chaîne de traitement du signal par un plan d'opérations primitives. Un opérateur primitif réalise un but simple et possède des méthodes d'initialisation et d'ajustement. Une composition d'opérateurs primitifs constitue un opérateur complexe. Un traitement est représenté par une requête décomposable en sous buts. Pour chaque requête, le plan de sous buts est exécuté puis adapté par un module de contrôle de l'exécution. Cette approche, bien que posant les bases de l'architecture d'un système de traitement du signal, est surtout proposée pour l'aide à la création d'une application et non pour son adaptation en ligne. De plus, leur méthode implique une séparation nette entre les connaissances purement traitement du signal et celles du domaine médicale et ne permet pas leur fusion pour prendre une décision. Karsai & Sztipanovits (Karsai & Sztipanovits, 1999) proposent une architecture pour les programmes structurellement auto-adaptifs en ligne. Le plan du traitement à effectuer est représenté par un graphe dont les nœuds sont les opérations primitives et dont les arcs représentent les flux de signaux. Le plan est exécuté sous le contrôle d'un planifieur séparé qui est configuré par un graphe de contrôle. Un opérateur complexe est représenté par une composition d'opérateurs primitifs sous forme de graphe. Cet *opérateur composé* contient plusieurs graphes de connexion alternatifs stockés dans les états d'une machine à états finis. Les transitions entre les états sont provoquées par des événements (alarmes) qui entraine le changement de plan, c'est-à-dire le changement d'architecture de l'opérateur composé.

Nous nous inspirons de ces approches pour piloter le système de monitoring Calicot. Dans notre étude, l'abstraction temporelle est décomposée en *tâches* inter-connectées. Tout comme un opérateur composé, une tâche peut être réalisée par un ensemble d'algorithmes. Cet ensemble d'algorithmes est modifié en ligne par le pilote en fonction des informations courantes du système, rassemblées sous le nom de contexte courant. Les tâches peuvent aussi être désactivées en fonction de la possibilité technique de leur réalisation, ce qui nécessite une adaptation du niveau de détail avec lequel le diagnostic médical est effectué. Contrairement aux approches présentées, notre pilote centralise les informations puis déduit l'ensemble des modifications à effectuer sur la chaîne de traitement.

Après une introduction aux arythmies cardiaques et au système de monitoring Calicot en section 2, les motivations de notre étude sont explicitées en section 3. Puis, la

nouvelle architecture et les bases de connaissance sont décrites en section 4. Le moteur d'inférences du pilote avec ses règles de pilotage sont ensuite détaillés en section 5. Enfin, les résultats du pilotage d'une tâche de détection sont présentés en section 6.

2 Présentation du contexte de l'étude

Nous développons un système de surveillance de patients qui présentent des *arythmies cardiaques*. C'est une évolution du système Calicot que nous détaillons après une brève introduction aux arythmies cardiaques.

2.1 Les arythmies cardiaques et l'électrocardiographie

Avant d'introduire les arythmies cardiaques, il est nécessaire de rappeler quelques généralités sur le cœur. Le cœur est composé de quatre chambres — deux oreillettes et deux ventricules — qui communiquent entre elles par un système de valves. L'oreillette de la partie droite (resp. gauche) recueille le sang des veines du corps (resp. des poumons) et le communique au ventricule droit (resp. gauche) qui l'expulse vers les poumons (resp. les artères du corps). Ce mécanisme est assuré par un stimulus électrique qui parcourt le cœur et commande la contraction de chaque chambre de manière adéquate. On appelle arythmie toute perturbation dans le parcours de ce stimulus par rapport à la normale. Le stimulus peut être mesuré en plaçant des électrodes à la surface du corps en des endroits prédéfinis. On utilise couramment en clinique l'électrocardiogramme (ECG) de surface pour obtenir une image de la propagation du stimulus à travers le cœur.

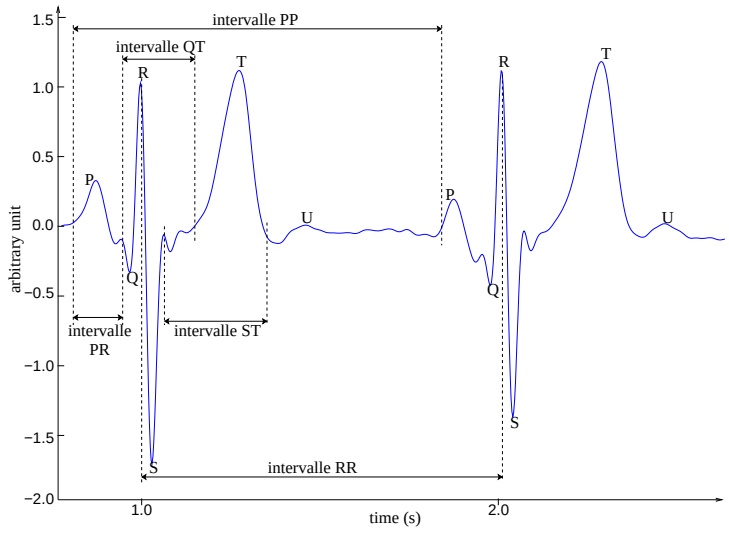


FIG. 1 – Ondes et intervalles standard d'un électrocardiogramme normal.

Le tracé de la figure 1 représente les ondes et intervalles standards étudiés en électrocardiographie. Les ondes P et QRS correspondent respectivement à la dépolarisation des oreillettes et des ventricules qui induit la contraction de ces chambres. L'onde T correspond à la repolarisation des ventricules.

Les arythmies peuvent être diagnostiquées à partir de la forme des ondes P et QRS et de leur écart temporel. Par exemple, un rythme *rapide* et régulier de QRS en forme d'*extrasystole ventriculaire* (commande prématurée de contraction d'un ventricule) permet le diagnostic de tachycardie *ventriculaire* alors qu'un rythme *rapide* et régulier de QRS *normaux* précédés d'*ondes P* indique une tachycardie *supra-ventriculaire*. Le contexte du patient (médicaments, âge, pathologies, etc.) doit aussi être pris en compte pour le diagnostic.

2.2 Le système de monitoring Calicot

Calicot (Cardiac Arrhythmias Learning for Intelligent Classification of On-line Tracks) (Carrault *et al.*, 2003) est un système de reconnaissance et d'apprentissage d'arythmies cardiaques. Sa principale caractéristique est d'associer trois domaines distincts : l'apprentissage symbolique, le traitement du signal et la reconnaissance de chroniques. Les arythmies sont vues comme *des événements temporellement liés entre eux*, ce qui est la définition même d'une chronique (Cordier & Dousson, 2000). Succinctement, Calicot est composé de trois principaux modules : un module d'*apprentissage des arythmies cardiaques*, un module d'*abstraction temporelle*, et un module de *reconnaissance de chroniques* (voir figure 2).

L'apprentissage des arythmies cardiaques (hors ligne)

L'apprentissage des arythmies cardiaques est réalisé par programmation logique inductive, à partir d'exemples positifs et négatifs d'ECG d'arythmies annotés. L'apprentissage produit des règles de reconnaissance qui sont traduites en modèles de chroniques. Ainsi, à chaque arythmie correspond au moins un modèle de chronique.

L'abstraction temporelle (en ligne)

Ce module transforme un signal électrocardiographique numérique en événements temporels étiquetés (complexe QRS normal ou anormal, onde P). L'abstraction temporelle est composée d'algorithmes de traitement du signal qui détectent et classent les événements de l'ECG.

La reconnaissance de chroniques (en ligne)

À partir des modèles de chroniques, ce module réalise la reconnaissance des chroniques (*i.e.* des arythmies cardiaques) dans le flux d'événements générés par l'abstraction temporelle. C'est cette partie qui réalise le *diagnostic médical*.

Calicot présente des performances satisfaisantes sur les exemples d'arythmies et de signaux traités (Carrault *et al.*, 2003). Cependant, le système reste sensible aux erreurs de l'abstraction temporelle qui peuvent perturber la reconnaissance des arythmies.

3 Introduction du pilotage dans Calicot

Nous partons du principe que les informations de l'analyse du signal et de la reconnaissance d'arythmies en cours permettent l'amélioration de la chaîne de traitement. Ceci nous a amené à introduire un pilote pour effectuer les modifications de Calicot en fonction de ces informations. La structure de Calicot autorise un pilotage à trois niveaux : au niveau de l'activation des tâches, au niveau de la sélection de la précision de la reconnaissance d'arythmies et au niveau de l'adaptation des algorithmes de traitement du signal.

Le pilotage des tâches

L'abstraction temporelle de Calicot peut être décomposée en quatre tâches principales :

Filtering : la tâche de filtrage sépare au maximum le signal ECG des bruits parasites,
QRSDetection : la tâche de détection des QRS donne leur date d'occurrence,
QRSClassification : la tâche de classification des QRS les étiquette, et
PWaveDetection : la tâche de détection des onde P donne leur date d'occurrence.

Dans Calicot, chaque tâche est constamment activée. Or, selon les cas, certaines tâches ne peuvent pas être accomplies correctement. Par exemple, lorsque la ligne est trop bruitée, la détection de l'onde P est vouée à l'échec et entraîne beaucoup d'erreurs. Dans ce contexte, cette tâche est pénalisante car elle fournit de fausses informations à la reconnaissance de chroniques. Pour fonder le diagnostic sur des informations fiables, une amélioration consiste à activer et désactiver les tâches en fonction du contexte.

Le pilotage de la reconnaissance d'arythmies

Le pilotage des tâches implique un pilotage de la reconnaissance d'arythmies. En effet, même si certaines tâches ne peuvent être activées, le diagnostic médical doit pourtant être établi. Par exemple, lorsque l'on se trouve en présence d'un rythme cardiaque rapide avec des QRS rapprochés, on peut être en présence de deux arythmies : une tachycardie ventriculaire (pouvant dégénérer en fibrillation mortelle) et une tachycardie supra-ventriculaire (moins dangereuse). Lorsque l'abstraction temporelle est très précise (toutes les tâches actives), on étudie la présence ou non d'ondes P pour les discriminer. Mais, lorsque les ondes P ne sont pas techniquement détectables, le système peut utiliser la forme des QRS pour distinguer les deux arythmies. Pour représenter ces différentes stratégies de diagnostic, plusieurs jeux de modèles de chroniques (*i.e.* modèles d'arythmies) sont appris en fonction du niveau de détail de description de l'ECG. Le pilote a pour rôle de sélectionner les modèles de chronique correspondant au niveau de détail de l'abstraction temporelle (tâches actives), autrement dit, en fonction du contexte courant.

Le pilotage des algorithmes de traitement du signal

Dans l'abstraction temporelle, chaque tâche est réalisée par un ou plusieurs algorithmes de traitement du signal inter-connectés. Or, dans la littérature, pour réaliser une tâche, il existe plusieurs algorithmes dont les performances varient en fonction du

contexte d'utilisation. Portet *et al.* (Portet *et al.*, 2005) ont montré que les performances de différents détecteurs de QRS varient en fonction du contexte courant. Ainsi, le pilotage de l'abstraction temporelle choisit les algorithmes et leurs paramètres les plus adaptés aux tâches à accomplir en fonction du contexte courant.

Un pilotage efficace est donc entièrement dépendant de la détection précise du contexte courant que nous allons décrire dans la section suivante.

4 Le système de pilotage

L'architecture de notre système intègre des détecteurs de contexte et un pilote. Les performances du pilotage sont dépendantes de la détection du contexte. Nous introduisons tout d'abord notre représentation du contexte courant puis l'architecture générale, les bases de connaissance et l'architecture du pilote.

4.1 Le contexte courant et sa détection

Le contexte courant est composé de trois sous-contextes : le *contexte de ligne*, le *contexte arythmique*, et le *contexte patient*. Le contexte patient (âge, rythme ECG de base, etc.) est statique alors que le contexte de ligne et le contexte arythmique sont dynamiques et remis à jour régulièrement par deux détecteurs.

Le contexte de ligne

Le contexte de ligne est constitué du niveau et du type de bruit présent sur la ligne à l'instant n. Nous reprenons de Portet *et al.* (Portet *et al.*, 2005) les trois types de bruit mutuellement exclusifs (bw, ma, et em) pour trois rapports signal sur bruit (5, -5, et -15 dB). Le bruit bw (baseline wander) est majoritairement basse fréquence, le bruit ma (muscle artefact) est majoritairement haute fréquence, et le bruit em (electrode motion artefact) a des composantes hautes et basses fréquences. La ligne peut, bien sûr, ne présenter aucun bruit. Le contexte prend donc une valeur parmi dix :

$$\forall\, n \in \mathbf{N},\ ctxtline(n) \in (\{bw, ma, em\} \times \{5, -5, -15\}) \cup \{no\_noise\}$$

Le détecteur de contexte de ligne est basé sur un détecteur classique de formes. Comme il est placé au début de la chaîne, il communique rapidement au pilote le contexte de ligne courant. Le pilote peut alors modifier l'abstraction temporelle avant que l'ECG ne soit traité.

Le contexte arythmique

Le contexte arythmique est constitué de la liste des arythmies en cours de reconnaissance. Le détecteur de contexte arythmique utilise les hypothèses du module de reconnaissance de chroniques afin d'établir une liste des arythmies les plus susceptibles d'apparaître. Cela permet au pilote de faire des hypothèses sur les formes d'ondes que l'abstraction temporelle doit traiter. Les principales formes de QRS sont symbolisées ici

par les lettres N, V, L, R, J, F, A, P, correspondant respectivement à battement Normal, extrasystole Ventriculaire, bloc de branche gauche (Left), etc.

4.2 La nouvelle architecture

L'architecture de notre système complète celle de Calicot. Le pilotage et les éléments nécessaires à son fonctionnement s'insèrent dans la chaîne de traitement. La figure 2 illustre les connexions entre les composants de notre système.

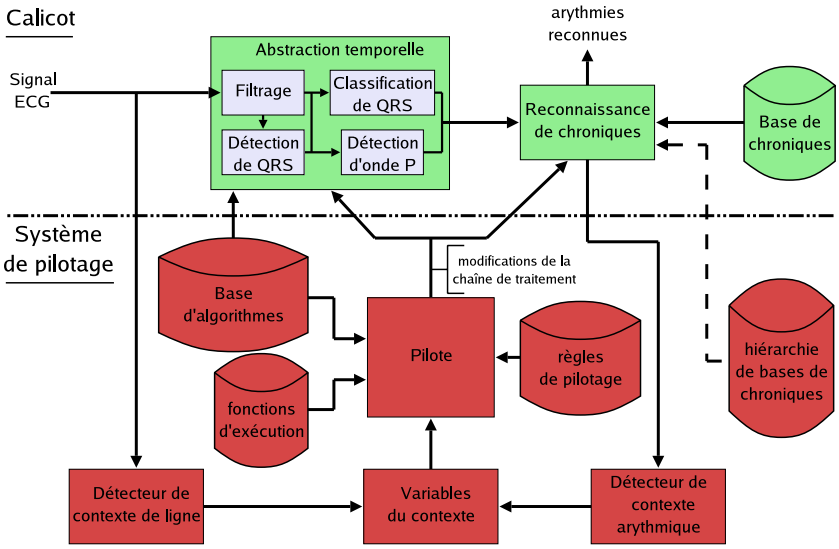


FIG. 2 – Architecture générale du système. La chaîne du haut représente le système Calicot et celle du bas représente les modules du système de pilotage.

Les deux détecteurs de contexte alimentent les variables du contexte. Le pilote effectue les modifications de la chaîne de traitement grâce aux variables du contexte et aux bases de règles de pilotage, d'algorithmes et de fonctions d'exécution. La base d'algorithmes contient un ensemble d'algorithmes de traitement du signal qui peuvent effectuer les tâches de l'abstraction temporelle. La base de fonctions d'exécution contient un ensemble de fonctions qui effectuent les modifications décidées par le pilote.

4.3 Les bases de connaissances

Le pilote intervient sur le système grâce à quatre bases de connaissances : une base d'algorithmes, une base de fonctions d'exécution, une base de chroniques et une base de règles de pilotage. Cette dernière est présentée avec le moteur d'inférences du pilote en section 5.

La base d'algorithmes

La base d'algorithmes contient tous les algorithmes utilisés dans Calicot pour la réalisation des tâches de l'abstraction temporelle : *Filtering*, *QRSDetection*, *QRSClassification* et *PWaveDetection*. Dans cette étude, nous nous focalisons sur la tâche *QRSDetection* pour laquelle trois algorithmes sont pilotés :

pan : le Pan et Tompkins (Pan & Tompkins, 1985) qui est un algorithme devenu standard dans la détection du QRS,

$gritzali$: le détecteur de Gritzali (Gritzali, 1988), autre algorithme standard de détection de QRS, et

$df2$: le détecteur de Okada modifié par Friesen *et al.* (Friesen *et al.*, 1990).

La base de fonctions d'exécution

La base de fonctions d'exécution permet de réaliser la modification de la chaîne de traitement. Elles sont de trois types :

$start(task)$: la tâche $task$ est activée, elle génère des événements qu'elle envoie à la reconnaissance d'arythmies,

$stop(task)$: la tâche $task$ est désactivée, elle n'envoie plus d'événements,

$set(task, parameter)$: la tâche $task$ est reparamétrée.

Les actions $start$ et $stop$ raniment ou mettent en veille la tâche afin de garder ses valeurs internes. Lorsqu'une tâche est en veille, elle n'effectue aucun traitement. Chaque tâche contient une méthode set capable de reparamétrer sa chaîne de traitement en fonction des paramètres passés.

La base de chroniques

Lorsqu'une tâche de l'abstraction temporelle est désactivée, l'ECG est analysé de manière moins précise. Les niveaux de détail de description de l'ECG sont représentés par une hiérarchie de langages de description :

– L1 inclut les dates d'occurrence des QRS et l'écart temporel entre QRS,
– L2 ajoute à L1 les types (morphologies) de QRS,
– L3 ajoute à L1 les dates d'occurrence des ondes P, et
– L4 ajoute à L2 les dates d'occurrence des ondes P.

Les chroniques ont été apprises avec des exemples décrits par les quatre langages pour générer une hiérarchie de modèles de chroniques *C1, C2, C3 et C4*. Ainsi, pour chaque combinaison de tâches actives, il existe une base de chroniques au niveau de détail correspondant. Par exemple, si seule la tâche *QRSDetection* est active alors il faut choisir la base de chroniques *C1*.

4.4 Le pilote

La structure du pilote est représentée par la figure 3.

Le pilote contient un *moteur d'inférences*, qui déduit les actions à effectuer sur la chaîne de traitement en fonction des variables du contexte, de la base de règles et aussi de son *contexte interne*. Le contexte interne du pilote contient les informations sur

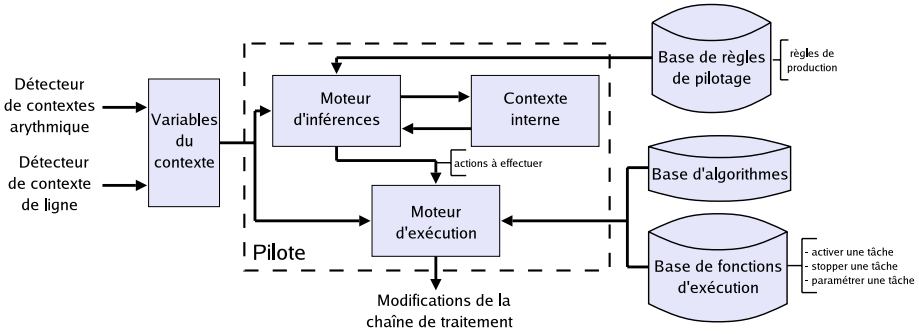


FIG. 3 – Architecture du pilote.

les tâches actives et les dernières modifications effectuées. En effet, l'abstraction temporelle de Calicot est vue comme la réalisation de plusieurs tâches interdépendantes. L'activation de ces tâches est dépendante du contexte et des autres tâches en cours. Par exemple, la tâche *QRSClassification* peut être activée uniquement si la tâche *QRSDetection* est active en même temps. C'est pourquoi, il faut mémoriser dans un contexte interne les dernières modifications effectuées sur le système.

Une fois les actions à effectuer déduites, le *moteur d'exécution* est chargé de la modification de la chaîne de traitement. Le moteur utilise des fonctions d'exécution qui permettent une gestion souple de chaque tâche. En effet, certaines tâches possèdent des variables internes à mettre à jour continuellement. Par exemple, la tâche *QRSDetection* possède une variable correspondant à la date courante. Si cette tâche est activée et désactivée sans précaution les données internes pourraient être perdues.

5 Le moteur d'inférences

Le moteur d'inférences est le cœur du pilote. Il déduit les actions à effectuer sur le système par un raisonnement qui utilise des informations du domaine du traitement du signal (tel que le contexte de ligne) et des informations du domaine médical (tel que le contexte arythmique). Sa connaissance est constituée du contexte courant et d'un ensemble de règles de pilotage représentées sous forme de règles de production.

SI

$<(A \wedge B) \vee (C \wedge D)>$

ALORS

<faire action i>

5.1 Le raisonnement du pilote

Le raisonnement du moteur d'inférences imite le raisonnement d'un expert ayant des connaissances médicales et techniques (traitement du signal). Le point de vue architec-

tural du raisonnement est illustré par la figure 4.

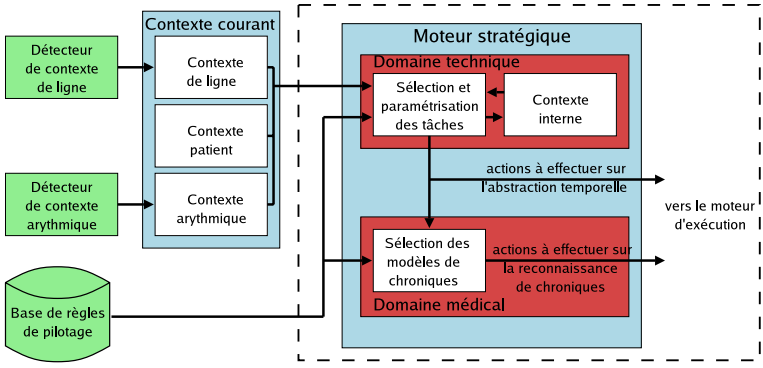


FIG. 4 – Le moteur d'inférences du pilote.

Dans le domaine technique, l'*expert* se sert du contexte courant mis à jour et de ses règles de pilotage pour déduire les tâches techniquement possibles et les meilleurs algorithmes pour les réaliser. Il infère les actions à effectuer pour modifier l'abstraction temporelle. Ensuite, il choisit la base de modèles de chroniques la plus adaptée en fonction des tâches possibles, autrement dit, en fonction de la précision de l'abstraction temporelle. Enfin, l'ensemble des actions à effectuer (changement de modèles, changement d'algorithmes, etc.) est transmis au moteur d'exécution. Par exemple, pour distinguer une tachycardie ventriculaire et une tachycardie supra-ventriculaire, la démarche intuitive est de choisir une stratégie médicale basée sur l'onde P. Si les ondes P ne sont pas techniquement détectables (trop de bruit), la reconnaissance sera erronée. Mais, un expert possédant des connaissances techniques pourra choisir une stratégie basée uniquement sur le QRS. Si une tâche est techniquement possible, le raisonnement technique décidera quels algorithmes et quels paramètres choisir grâce aux variables du contexte. Par exemple, si la tâche de détection de QRS est activée alors que la ligne présente du bruit em, le détecteur $df2$ sera choisi plutôt que le détecteur pan.

5.2 Les règles de pilotage

Les règles de pilotage sont principalement définies par un expert et sont regroupées en règles techniques de fonctionnement, en règles de choix de tâches, en règles de choix de modèles de chroniques, et en règles de pilotage des algorithmes de traitement du signal. Nous détaillons ces classes de règles avec des exemples.

Règles techniques de fonctionnement

Les règles techniques de fonctionnement mettent à jour des variables utiles pour la suite du pilotage. Par exemple, la règle suivante permet d'initialiser une variable pour désactiver la tâche de détection d'ondes P.

SI

$<$*ctxtligne* $!=$ *no_noise*$>$

ALORS

$<$*tooMuchNoiseForPWave*$>$

Règles d'activation des tâches

Les tâches sont activées principalement selon le contexte de ligne, par exemple pour activer *PWaveDetection* il est nécessaire d'avoir une ligne non bruitée.

SI

$<\neg$*Active(PWaveDetection)*$\wedge\neg$*tooMuchNoiseForPWave*$>$

ALORS

$<$*start(PWaveDetection)*$>$

S'il est techniquement possible d'obtenir la tâche de détection d'ondes P, alors la tâche et activée mais seulement si elle n'était pas déjà active. Après cette activation, le prédicat *Active(PWaveDetection)* devient vrai. Ce type de règle existe pour chaque tâche.

Règles de choix du niveau de détail de la reconnaissance de chroniques

Les modèles de chroniques doivent être adaptées à la précision de l'abstraction temporelle. Par exemple, si les tâches *QRSDetection*, *QRSClassification* sont actives et *PWaveDetection* est inactive, alors le reconnaisseur de chroniques doit utiliser la base de chroniques *C2*.

SI

$<$*Active(QRSDetection)* $\wedge$ *Active(QRSClassification)* $\wedge$
$\neg$*Active(PWaveDetection)*$>$

ALORS

$<$*utiliser la base de chroniques C2*$>$

Règles de pilotage d'algorithmes de traitement du signal

Après le choix des tâches, les règles de pilotage d'algorithmes permettent de les paramétrer. Par exemple, si la tâche *Active(QRSDetection)* est active il faut choisir le détecteur le plus approprié :

Pour la première règle, si le contexte de ligne dit qu'il y a présence de bruit bw à -5 dB et que le contexte médical informe qu'il a majoritairement des QRS de forme L, alors le détecteur $gritzali$ est sélectionné. Ces règles, plus riches que les précédentes, sont dérivées de Portet *et al.* (Portet *et al.*, 2005).

SI

$<L \wedge bw \wedge SNR \geq -5dB>$

ALORS

<set(QRSDetection,changeQRS(gritzali,param(gritzali,ctxtligne)))>

SI

<(L∨F) ∧ no_noise>

ALORS

<set(QRSDetection,changeQRS(gritzali,param(gritzali,ctxtligne)))>

SI

$<(F \vee P) \wedge bw \wedge SNR \geq 0dB>$

ALORS

<set(QRSDetection,changeQRS(gritzali,param(gritzali,ctxtligne)))>

SI

$<em \wedge ((N \vee A \vee P \vee R) \wedge SNR = -15dB)>$

ALORS

<set(QRSDetection,changeQRS(df2,param(df2,ctxtligne)))>

SI

$<em \wedge (SNR = -5dB \wedge P)>$

ALORS

<set(QRSDetection,changeQRS(df2,param(df2,ctxtligne)))>

SI

<default>

ALORS

<set(QRSDetection,changeQRS(pan,param(pan,ctxtligne)))>

6 Résultats du pilotage de la tâche de détection du QRS

Le pilotage d'algorithmes a été testé sur cinq ECG générés à partir de la base standard MIT-BIH arrhythmia database[1]. Ces ECG ont été bruités avec du bruit clinique réel de type bw, ma et em et le pilotage s'est effectué avec des détecteurs parfaits de contexte. Les ECG ont une durée de 20 à 30 minutes et représentent au total 2 heures d'enregistrement. Ils sont composés des contextes suivants :

- `ECG_1` : 1200 QRS dont 300 de forme L sans bruit, 300 de forme V avec bw à -5dB, 300 de forme P avec bw à -5dB et 300 de forme V avec bw à -5dB
- `ECG_2` : 1800 QRS de forme L dont 300 bruitées ma à -5dB, 300 sans bruit, 600

[1] http ://ecg.mit.edu/

bruités bw à -5dB et 600 sans bruit
- `ECG_3` : 1200 QRS de forme N dont 300 sans bruit, 300 bruités em à -15dB, 300 sans bruit et 300 bruités em à -15dB
- `ECG_4` : 1200 QRS de forme R dont 300 sans bruit, 300 bruités em à -15dB, 300 sans bruit et 300 bruités em à -15dB
- `ECG_5` : 1800 QRS de forme P dont 600 sans bruit, 600 bruités em à 5dB et, 600 bruitées em à -5dB

Chaque ECG est composé de 3 à 4 contextes différents de façon à évaluer non seulement, les performances des détecteurs dans ces contextes, mais aussi, le comportement du pilotage au moment des transitions entre contextes. Pour chacun de ces contextes, le pilote décide, avec les règles de pilotage, quel algorithme est le plus adapté. Pour chaque test, le nombre de faux négatifs FN (QRS non détectés) et de faux positifs FP (fausses alarmes) sont calculés pour obtenir le nombre d'erreurs, $Ne = FP + FN$. Le taux d'erreurs est $Te = \frac{Ne}{N_{QRS}}$ où N_{QRS} représente le nombre total de QRS présents. Le seuil des détecteurs est choisi de manière optimale dans le sens où Ne est minimum. Les détecteurs sont ainsi utilisés au maximum de leur performance.

TAB. 1 – résultats du pilotage de la tâche de détection du QRS

ECG	1	2	3	4	5	total	
	Ne	Ne	Ne	Ne	Ne	Ne	$Te(\%)$
pan	**20**	***91**	*240	*312	*367	**1030**	**14,3**
$gritzali$	**20**	*160	388	360	***295**	1223	17
$df2$	307	278	***174**	***160**	*302	1221	17
$pilote$	20	88	185	167	304	764	10,6

***** algorithme utilisé par le pilote

Le tableau 1 montre que le meilleur algorithme (en gras) est différent pour chaque ECG. Le pilotage obtient toujours un score très proche du détecteur optimal car il tire parti, dans chaque contexte, des performances du meilleur algorithme. Dans le cas des signaux `ECG_3`, `ECG_4` et `ECG_5`, les résultats du pilote sont un peu moins bons que l'utilisation simple de $df2$. Cette différence est due aux transitions de contextes. En effet, pour être traité en temps réel, le signal est découpé en buffer. Or, même si un buffer contient deux contextes différents (transitions entre contextes), il sera traité par un seul détecteur. Par exemple dans `ECG_3`, la transition entre le premier contexte sans bruit et le deuxième contexte avec em à -15dB est contenu dans un buffer et le choix du pilote a été de choisir pan alors que $df2$ aurait été moins pénalisant. Cependant, le pilotage permet de diminuer de façon conséquente le nombre d'erreurs total générées par rapport à l'utilisation simple du meilleur détecteur en moyenne (ici pan). Ces résultats montrent l'intérêt d'utiliser un pilotage intelligent des algorithmes de détection de QRS en utilisant une fusion d'informations du domaine du traitement du signal (niveau de bruit) et du domaine médical (forme des QRS). Ceci ouvrent des perspectives encourageantes pour le pilotage des autres tâches de l'abstraction temporelle.

7 Conclusion et perspectives

Nous proposons un pilotage du système de monitoring Calicot. Ce pilotage choisit des algorithmes dans une base pour modifier en ligne la chaîne de traitement du signal. Il est guidé par la détection du contexte courant que nous définissons comme étant principalement constitué de l'analyse du bruit de ligne et du diagnostic médical. L'architecture proposée permet aussi de n'activer que les tâches fiables et d'adapter la reconnaissance d'arythmies à la précision courante du système. Les résultats montrent que le taux d'erreurs moyen de 14.3% obtenu avec un seul algorithme de détection de QRS passe à 10.6% avec le pilotage. Dans cette article, la précision de l'abstraction temporelle est guidée par les possibilités techniques de leur réalisation. Une autre partie de notre travail consiste à guider le pilotage de l'abstraction temporelle par la reconnaissance d'arythmies. En effet, si nous nous trouvons dans un contexte où il est possible de discriminer deux arythmies sans utiliser la détection d'ondes P, qui est très coûteuse en temps et peu robuste, alors le pilote n'activera pas cette tâche. Ce pilotage à deux niveaux, c'est-à-dire un pilotage qui prend en compte les besoins d'abstraction de la reconnaissance d'arythmies et la faisabilité des tâches de l'abstraction temporelle, fait partie de nos prochains travaux.

Références

CARRAULT G., CORDIER M., QUINIOU R. & WANG F. (2003). Temporal abstraction and inductive logic programming for arrhyhtmia recognition from electrocardiograms. *Artificial Intelligence in Medicine*, **28**, 231–263.

CORDIER M. & DOUSSON C. (2000). Alarm driven monitoring based on chronicles. In *Safeprocess'2000*, p. 286–291.

FRIESEN G. M., JANNETT T. C., JADALLAH M. A., YATES S. L., QUINT S. R. & NAGLE H. T. (1990). A comparison of the noise sensitivity of nine QRS detection algorithms. *IEEE Transactions on Biomedical Engineering*, **37**, 85–98.

GRITZALI F. (1988). Towards a generalized scheme for QRS detection in ECG waveforms. *Signal Processing*, **15**, 183–192.

KARSAI G. & SZTIPANOVITS J. (1999). A model-based approach to self-adaptive software. *IEEE Intelligent systems*, **14**, 46–53.

LARSSON J. & HAYES-ROTH B. (1998). Guardian : An intelligent autonomous agent for medical monitoring and diagnosis. *IEEE Intelligent Systems*, **13**, 58–64.

PAN J. & TOMPKINS W. J. (1985). A real-time QRS detection algorithm. *IEEE Transactions on Biomedical Engineering*, **BME-32(3)**, 230–236.

PORTET F., HERNÁNDEZ A. I. & CARRAULT G. (2005). Evaluation of real-time QRS detection algorithms in variable contexts (accepté). *Medical & Biological Engineering & Computing*.

SHEKHAR C., MOISAN S. & THONNAT M. (1994). Towards an intelligent problem-solving environment for signal processing. *Mathematics and Computers in Simulation*, **36**, 347–359.

SOULAS T., CERTEN G. L., PICHON J. L. & CARRAULT G. (1998). Algorithm switching in real time monitoring. In *Symposium on Electronics and Telecommunications (ETC)*, p. 145–149, Timisoara.

Une nouvelle approche de l'intégration de la planification de production et de l'ordonnancement

Anna Robert[1,2], Claude Le Pape[1], Frédéric Paulin[1], Francis Sourd[2]

[1] ILOG S.A., 9 rue de Verdun
94253 Gentilly cedex
{anrobert,clepape,fpaulin}@ilog.fr

[2] Laboratoire d'Informatique de Paris VI, 8 rue du Capitaine Scott
75015 Paris
francis.sourd@lip6.fr

Résumé : Dans le domaine de l'industrie de transformation, l'intégration de la planification de production et de l'ordonnancement est cruciale. Généralement des techniques hybrides de programmation linéaire, pour la planification, et de propagation de contraintes, pour l'ordonnancement, résolvent ce problème de manière coopérative. Nous nous proposons ici d'aborder cette question en résolvant un problème intermédiaire consistant en la découpe de la production en lots ordonnançables (*lot streaming*) et la détermination des flux de matériaux circulant entre les entités de la chaîne de production : inventaires, lots de production et demandes (*pegging*).

Mots-clés : Planification de production ; Ordonnancement ; Lot streaming ; Pegging ; Programmation linéaire en variables mixtes.

1 Introduction

Les problématiques de planification de production et d'ordonnancement ont été identifiées dès les années 60-70 avec l'émergence des MRP (*Material Requirements Planning*, Lunn & Neff (1992) et Hopp & Spearman (2000) par exemple). A l'heure actuelle, les problèmes liés à la gestion de production sont critiques pour les industriels, qui cherchent toujours à améliorer et optimiser leur rendement. Pour cela, il est nécessaire de concevoir des outils perfectionnés, adaptés à cette demande. Généralement, la planification se concentre sur des prévisions tactiques à long ou moyen terme de la production (de l'ordre de plusieurs mois) tandis que l'ordonnancement, se situant à un niveau de granularité plus fin, prend des décisions opérationnelles à court terme (quelques jours ou quelques semaines). Si l'on résout séparément ces deux problèmes, cela mène d'une part parfois à une incohérence entre la planification et l'ordonnancement (les solutions ne sont pas compatibles) et d'autre part, à des solutions globalement sous-optimales.

Une approche intégrée de ces problématiques semble donc une voie prometteuse tant du point de vue de la cohérence entre planification et ordonnancement, que de la qualité des solutions.

Généralement des techniques hybrides de programmation linéaire, pour la planification (problèmes de *lot sizing*, définition et classification dans Staggemeier & Clark (2001)), et de propagation de contraintes, pour l'ordonnancement, résolvent ce problème de manière coopérative afin d'aboutir effectivement à de meilleures solutions (par exemple Maravelias & Grossmann (2005) et Timpe (2002)). D'autres méthodes coopératives faisant appel à diverses techniques peuvent aussi être employées (Dauzère-Pérès & Lasserre (1994)). Nous proposons ici une approche résolvant un problème intermédiaire considérant les décisions de planification comme des données d'entrée et posant en sortie le problème d'ordonnancement. Elle consiste, d'une part, en une découpe optimale (au sens d'un objectif que nous définirons) de la production planifiée (*lot streaming*), et, d'autre part, en une prise de décisions relatives aux flux de matériaux circulant le long de la chaîne de production entre chacune des entités qui la composent (inventaires, lots de production, demandes), que nous désignerons par le terme *pegging*.

Dans une première partie, nous décrirons la problématique générale dans laquelle s'inscrivent nos travaux ainsi que l'approche particulière que nous envisageons pour répondre à la question qu'elle sous-tend. Nous présenterons dans un second temps un procédé de résolution pour notre problème consistant en une modélisation par programmation linéaire en variables mixtes que nous comparerons, dans une troisième partie, à une procédure naïve sur 54 instances.

2 Description du problème

2.1 Problématique globale

Afin d'exercer un meilleur contrôle sur leur activité, les industriels de la production souhaitent obtenir des informations à long et moyen termes, tout au long de la chaîne de production, les aidant à prendre de bonnes décisions, souvent cruciales. De plus, afin d'affiner ces prises de décision, des informations précises à plus court terme sont aussi nécessaires. Ces dernières permettent aux industries de faire preuve d'une forte réactivité afin qu'une petite défaillance - temporaire ou non - dans le système de production n'induise pas des complications en chaîne, en proposant rapidement des solutions alternatives. Nous sommes donc face à deux problèmes connus de la recherche opérationnelle : la **planification de production** et l'**ordonnancement**.

Nous commencerons par définir les concepts que nous manipulerons tout au long de cet article, puis nous donnerons une définition précise de ce que nous appelons planification de production et ordonnancement.

2.1.1 Concepts manipulés

Les notions qui interviennent explicitement dans notre problème sont les suivantes :

– Les **matériaux** : ils circulent tout au long de la chaîne de production ; ici, aucune distinction n'est faite entre matières premières, produits intermédiaires et produits

finis, un matériau peut appartenir à ces trois types.
- Les **inventaires** : ils représentent, pour chaque matériau, le stock de celui-ci.
- Les **recettes** : ce sont les principes de transformation des matériaux ; on associe à une recette un ensemble de matériaux consommés et un ensemble de matériaux produits.
- Les **lots** de production : un lot est associé à une recette et consiste en l'exécution d'une certaine quantité de celle-ci, ce qui induit une consommation et production de matériaux proportionnelles à cette quantité.
- Les **demandes** : il s'agit de quantités de matériaux requises par un client.
- Les **dates de livraison** : elles sont associées aux demandes.

Intervenant indirectement dans notre problème, le concept de **ressource** (coerrespondant généralement aux machines sur lesquelles s'exécutent les activités liées aux lots de production) est nécessaire à la description d'une usine.

2.1.2 Planification de production

Les décisions prises à moyen terme visent à équilibrer la production sur un ensemble de périodes donné (douze ou vingt-quatre mois, par exemple). La connaissance globale de ce type d'informations relève de la planification. Cette dernière prend des décisions sur les produits à fabriquer, le temps et la quantité de production, tout en respectant globalement des contraintes de satisfaction de demandes, de quantité et temps d'exécution des recettes bornés, de capacité des ressources et d'inventaire.

Dans notre modélisation du problème nous considérons qu'une solution au problème de planification contient, pour chaque période, les informations suivantes :
- la **quantité d'exécution** de chaque recette ;
- la **quantité de demande satisfaite** pour chaque demande ;
- pour chaque matériau, le **niveau d'inventaire** en fin de chaque période.

2.1.3 Ordonnancement

Dans un horizon à court terme, les ressources sur lesquelles on exécute la production interviennent de manière plus détaillée. Un ordonnancement des activités, correspondant aux lots de production, doit fournir les dates de démarrage et d'arrêt de la production de chaque matériau, ainsi que la ou les ressources nécessaires à cette production. Cette solution doit d'une part respecter avec exactitude les contraintes de capacité des ressources et satisfaire au mieux les dates de livraison des demandes, et d'autre part, dans le cas d'une approche intégrée, être autant que possible en accord avec les décisions prises par la planification.

Dans notre contexte, le problème d'ordonnancement consiste en la donnée d'un **ensemble de lots de production** à ordonnancer sur les ressources requises par les recettes exécutées, dans des **fenêtres de temps** induites par les périodes considérées par la planification, ainsi que des **contraintes de précédence** liées au rapport de production/consommation des lots.

2.2 L'approche *"lot streaming-pegging"*

Les quantités de production décidées lors de la planification ne peuvent être directement considérées comme des lots que l'on peut ordonnancer tels quels. En effet, ces lots sont de taille bornée (ceci est imposé par la description de la chaîne de production) et il convient donc de "découper" la quantité totale planifiée afin de satisfaire ces contraintes. Cette découpe est appelée *lot streaming* (Trietsch & Baker (1993), Baker & Jia (1993), Roux (1997) et bien d'autres). En outre, des décisions concernant le flux des matériaux sont prises avant l'ordonnancement. Ce second problème, que nous nommons *pegging*, est couplé au lot streaming pour une résolution simultanée sur chacune des périodes de temps induites par la planification (voir Fig. 1).

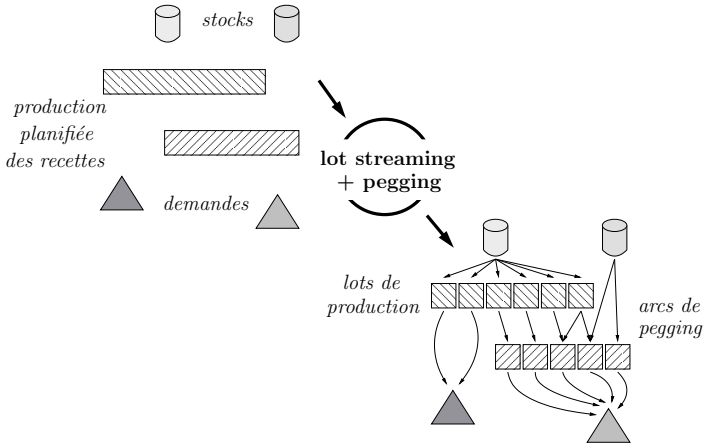


FIG. 1 – *De la planification au lot streaming/pegging*

2.2.1 Structure et position du problème d'optimisation

Dans la suite, nous nous placerons toujours au niveau d'une période de temps au sens de la planification.

Une structure double

Du fait que nous résolvons le lot-streaming et le pegging simultanément, le modèle formulant le problème général présente une structure particulière possédant deux sous-structures, chacune étant liée à l'un des sous-problèmes. Le lot-streaming peut être vu comme un problème de découpe et le pegging comme celui de l'établissement d'un flot entre les morceaux issus de cette découpe et d'autres entités extérieures, les deux sous-problèmes doivent donc être couplés par certaines contraintes.

Nous présentons d'abord les deux sous-problèmes et les contraintes dont relève chacun d'eux, puis, les contraintes qui permettent de relier ces deux structures, formant

ainsi le problème global ; enfin, nous présenterons la fonction objectif. Nous ferons au fur et à mesure référence à la formulation mathématique donnée en section 3.

Le lot streaming : un problème de découpe

Connaissant la quantité totale d'exécution de chaque recette (pièce à découper), le problème consiste à définir des lots (morceaux résultant de la découpe) dont la taille est soumise à différentes contraintes :
- bornes inférieure et supérieure (équations (5) et (6)),
- taille constante facultativement requise (équ. (7), (8) et (9)),
- respect de la taille totale de la pièce (équ. (15)).

Le pegging : un problème de flot

Ce problème consiste en la détermination, pour chaque lot, des parts de consommation provenant du stock et/ou de lots producteurs (pour chaque matériau consommé) et des parts de production allant en stock et/ou consommées par des lots consommateurs et/ou participant à la satisfaction de demandes (pour chaque matériau produit). Ce flux est représenté par des arcs reliant stocks, lots et demandes. Une solution de pegging contient donc des arcs liant ces entités, sur lesquels figure une quantité de matériau en circulation, et ce, tout au long de la chaîne de production. Nous distinguons 5 types d'arcs de pegging :
- Les **Stock2Lot** : entre stocks et lots consommateurs.
- Les **Lot2Lot** : entre lots producteurs et lots consommateurs.
- Les **Lot2Stock** : entre lots producteurs et stocks.
- Les **Stock2Dem** : entre stocks et demandes.
- Les **Lot2Dem** : entre lots producteurs et demandes.

Ce problème évoque le concept de flot, puisqu'en effet, nous sommes toujours en présence d'un réseau spécifiant les rapports de production/consommation entre des entités productrices et consommatrices, dont on détermine simultanément (par le lot streaming) la quantité d'exécution (ce qui s'apparente à une capacité).

Considérons que le lot streaming est déjà résolu. Si on est en présence de 2 recettes R1 et R2, R1 produisant un matériau M que consomme R2, si la décision du lot streaming est d'en exécuter 2 lots de chaque (R1/1 et R1/2, R2/1 et R2/2), chacun étant de taille 2, le flux des matériaux étant continu et pouvant être divisé, on a une infinité de solutions au problème consistant à décider du transfert de M entre ces 4 lots (voir Fig. 2).

Ce problème est soumis à des contraintes de conservation d'équilibre (ou contraintes de "flot") :
- consommation d'un lot (flux entrant, équ. (1)),
- production d'un lot (flux sortant, équ. (2)),
- niveau des stocks (respect de capacité, équ. (3)),
- satisfaction des demandes (respect de capacité, équ. (4)).

Couplage des deux structures

Le lien entre les deux problèmes composant le problème final est établi au moyen des contraintes de consommation et de production d'un lot (équ. (1) et (2)) qui font

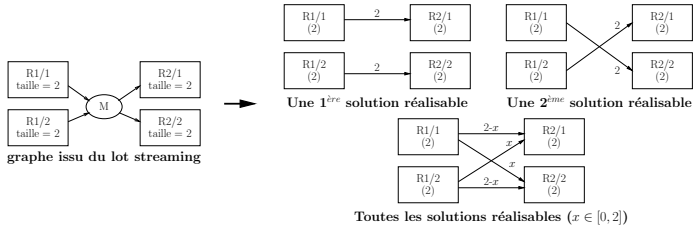


FIG. 2 – *Le pegging : un problème combinatoire*

intervenir la taille des lots (assimilable à une capacité) et le flux traversant ce lot.

Fonction objectif

Les dates de début et de fin de la période que l'on considère, les temps d'exécution des lots (se déduisant de leur taille par une relation affine) et la date prévue pour les satisfactions des demandes, permettent d'affecter des dates d'échéance "préférées" pour l'exécution de chaque lot. On peut donc calculer le délai entre les lots reliés par des arcs de type **Lot2Lot** (différence entre les deux dates d'échéance), et le délai entre les lots et demandes reliés par des arcs de type **Lot2Dem** (différence entre la date de livraison de la demande et la date d'échéance). Nous minimisons alors la somme sur tous les arcs de pegging de la valeur absolue de ces délais. En effet, il peut être nécessaire d'achever l'exécution d'un lot satisfaisant une demande après la date de livraison prévue pour celle-ci (afin de satisfaire d'autres contraintes plus fortes, comme les contraintes disjonctives de ressources), le délai est alors négatif.

2.2.2 Intérêts de l'approche

Apport général de la méthode

Une résolution simultanée des deux problèmes a la propriété de guider le lot streaming vers un découpage optimal dans le sens où si l'on détermine pour quel lot ou pour quelle demande tel lot va être exécuté (pegging) on peut adapter la taille de ce lot afin de ne produire ni trop (pour éviter des pénalités d'avance et/ou de stockage) ni trop peu (ce qui nécessiterait la mise en production d'un nouveau lot). Cette approche a donc aussi pour objectif de générer des ordonnancements de meilleure qualité, du point de vue de l'avance/retard.

D'un point de vue plus global, ce schéma présentant une structure double peut être étendu à d'autres problèmes d'optimisation, moyennant une adaptation du modèle et de l'objectif, comme le dimensionnement de réseaux spécialisés (les lots s'apparentant à des machines dont la puissance est à déterminer et le pegging, aux transferts de données), ou la gestion de l'exécution préemptive des processus sur une architecture mono ou multi-processeurs (les lots s'apparentent alors aux morceaux de processus et le pegging, aux transferts d'informations)...

Apports spécifiques du pegging

On constate que les décisions issues du pegging ne sont pas nécessaires pour poser un problème d'ordonnancement, seuls les lots et les fenêtres de temps le sont. Néanmoins, l'ajout d'arcs de pegging présente les avantages suivants :

- L'**introduction de nouvelles précédences** contraint le problème d'ordonnancement, ce qui, dans notre contexte où ce dernier est résolu par une méthode de propagation de contraintes, accélère la recherche.
- Des contraintes de consommation/production ($\mathcal{C}o\mathcal{P}r$) peuvent être ajoutées plus globalement au problème d'ordonnancement, indépendamment du pegging, celles-ci permettent de contrôler le niveau des stocks à tout instant mais sont consommatrices en temps de calcul ; les contraintes temporelles de pegging, présentant une **meilleure propagation** peuvent permettre, dans certains cas, de s'y substituer ou, dans d'autres, de s'y ajouter, et, à nouveau, d'accélérer la recherche, en écartant éventuellement la solution optimale du problème global.
- De manière plus informative, avoir des indications précises sur la façon dont se déroulent les opérations le long de la chaîne de production accroît le potentiel de réactivité de l'usine.

3 Présentation du modèle de résolution

Il s'agit d'un modèle mathématique de programmation linéaire en variables mixtes. Nous nous plaçons au niveau d'une période, pour laquelle nous avons les informations de planification et nous résolvons deux problèmes d'optimisation combinatoire : le lot streaming et le pegging. Une hypothèse est faite sur les données décrivant la chaîne de production : une recette ne peut à la fois consommer et produire un même matériau.

3.1 Notations

Nous regroupons dans le tableau 1 les notations employées dans les équations mathématiques décrivant notre modèle.

3.2 Programme linéaire en variables mixtes

3.2.1 Consommation/production

Ces contraintes de conservation d'équilibre concernent la production et la consommation des matériaux et ont été introduites dans la section 2.2.1. Les équations (1) (respectivement, (2)) expriment les contraintes de consommation (respectivement, production) pour chaque lot. Les équations (3) contraignent à respecter les niveaux d'inventaire de chaque matériau en début et fin de période. Enfin, les équations (4) forcent les demandes prévues à être satisfaites dans leur intégralité.

indice	**ensemble**	**concept désigné**
r	$\mathcal{R}$	recette
j	$\{1, \ldots, max_r\}^\star$	$j^{ème}$ lot de r
m	$\mathcal{M}$	matériau
m	$\mathcal{C}_r$/$\mathcal{P}_r$	matériau consommé/produit par r
d	$\mathcal{D}$	demande

$^\star max_r$ désigne le nombre maximal de lots possibles pour r

constante	**domaine**	**concept désigné**
δ_r^m	$\mathbb{R}^+$	proportion de consommation ou production de m par r ($m \in \mathcal{C}_r \cup \mathcal{P}_r$)
q_d	$\mathbb{R}^+$	quantité de matériau délivrée pour d
I_i^m/I_f^m	$\mathbb{R}^+$	quantité de m en stock en début/fin de période
$\underline{LS}_r$/$\overline{LS}_r$	$\mathbb{R}^+$	taille minimale/maximale d'un lot pour r
Q_r	$\mathbb{R}^+$	quantité d'exécution de r pour la période
α_r/β_r	$\mathbb{N}$/$\mathbb{R}^+$	durée fixe/variable d'exécution d'un lot de r
t_d	$\mathbb{N}$	date de livraison de d
$Start$/End	$\mathbb{N}$	date de début/fin de la période en cours

variable	**domaine**	**concept désigné**
x_{rj}	$\{0,1\}$	$= 1$ si le $j^{ème}$ lot de r est créé, 0 sinon
q_{rj}	$\mathbb{R}^+$	taille du $j^{ème}$ lot de r
X_{Irj}^m/Q_{Irj}^m	$\{0,1\}$/$\mathbb{R}^+$	$= 1$ s'il existe un **Stock2Lot** entre le stock de m et le $j^{ème}$ lot de r/quantité associée
X_{Id}/Q_{Id}	$\{0,1\}$/$\mathbb{R}^+$	$= 1$ s'il existe un **Stock2Dem** entre le stock du matériau requis par d et d/quantité associée
$X_{rjr'j'}^m$/$Q_{rjr'j'}^m$	$\{0,1\}$/$\mathbb{R}^+$	$= 1$ s'il existe un **Lot2Lot** entre le $j^{ème}$ lot de r et le $j'^{ème}$ lot de r', pour m/quantité associée
X_{rjd}/Q_{rjd}	$\{0,1\}$/$\mathbb{R}^+$	$= 1$ s'il existe un **Lot2Dem** entre le $j^{ème}$ lot de r et d/quantité associée
X_{rjI}^m/Q_{rjI}^m	$\{0,1\}$/$\mathbb{R}^+$	$= 1$ s'il existe un **Lot2Stock** entre le $j^{ème}$ lot de r et le stock de m/quantité associée
t_{rj}	$\mathbb{N}$	date de fin d'exécution du $j^{ème}$ lot de r
Z_{rjd}	$\mathbb{N}$	$= t_{rj}$ s'il existe un **Lot2Dem** entre le $j^{ème}$ lot de r et d, 0 sinon
$ZP_{rjr'j'}^m$	$\mathbb{N}$	$= t_{rj}$ s'il existe un **Lot2Lot** entre le $j^{ème}$ lot de r et le $j'^{ème}$ lot de r', 0 sinon
$ZC_{rjr'j'}^m$	$\mathbb{N}$	$= t_{r'j'}$ s'il existe un **Lot2Lot** entre le $j^{ème}$ lot de r et le $j'^{ème}$ lot de r', 0 sinon

TAB. 1 – MIP *lot streaming - pegging* : indices, ensembles, constantes, variables de décision

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \forall m \in \mathcal{C}_r,$$

$$Q^m_{Irj} \quad + \quad \sum_{\substack{r' \in \mathcal{R} \setminus \\ m \in \mathcal{P}_{r'}}} \sum_{j'=1}^{max_{r'}} Q^m_{r'j'rj} = \delta^m_r \times q_{rj} \quad (1)$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \forall m \in \mathcal{P}_r,$$

$$Q^m_{rjI} + \sum_{\substack{r' \in \mathcal{R} \setminus \\ m \in \mathcal{C}_{r'}}} \sum_{j'=1}^{max_{r'}} Q^m_{rjr'j'} \quad + \quad \sum_{\substack{d \in \mathcal{D} \setminus \\ m \text{ requis} \\ \text{par } d}} Q_{rjd} = \delta^m_r \times q_{rj} \quad (2)$$

$$\forall m \in \mathcal{M}, \quad \sum_{\substack{r \in \mathcal{R} \setminus \\ m \in \mathcal{C}_r}} \sum_{j=1}^{max_r} Q^m_{Irj} + \sum_{\substack{d \in \mathcal{D} \setminus \\ m \text{ req.} \\ \text{par } d}} Q_{Id} \quad - \quad \sum_{\substack{r \in \mathcal{R} \setminus \\ m \in \mathcal{P}_r}} \sum_{j=1}^{max_r} Q^m_{rjI} = I^m_i - I^m_f \quad (3)$$

$$\forall d \in \mathcal{D}, \quad Q_{Id} \quad + \quad \sum_{\substack{r \in \mathcal{R} \setminus \\ m, \text{ req.} \\ \text{par } d, \in \mathcal{P}_r}} \sum_{j=1}^{max_r} Q^m_{rjd} = q_d \quad (4)$$

3.2.2 Taille des lots

A nouveau, ces contraintes ont été décrites dans la section 2.2.1. Dans tous les cas (taille de lot constante ou non), la taille des lots est bornée, ceci est exprimé par les équations (5) et (6). Dans le cas particulier où une taille constante est requise pour les lots, on calcule leur taille par des équations linéaires (7), dans ces équations, on a toujours finalement $q_r = \frac{Q_r}{\text{nombre de lots}}$. Ainsi, on force les q_{rj} à prendre cette valeur par les contraintes (8) et (9).

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \qquad q_{rj} \geq \underline{LS}_r \times x_{rj} \quad (5)$$

$$q_{rj} \leq \overline{LS}_r \times x_{rj} \quad (6)$$

⋆ cas des tailles de lot constante :

$$\forall r \in \mathcal{R}, \qquad q_r = Q_r + \sum_{j=2}^{max_r} \left(\left(\frac{Q_r}{j} - \frac{Q_r}{j-1} \right) \times x_{rj} \right) \quad (7)$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \qquad q_{rj} \geq q_r - \overline{LS}_r \times (1 - x_{rj}) \quad (8)$$

$$q_{rj} \leq q_r \quad (9)$$

3.2.3 Précédence temporelle

Lorsque deux lots sont liés par un arc de pegging, ils sont contraints d'être exécutés l'un après l'autre, le lot consommateur s'exécutant nécessairement après que le produc-

teur ait achevé son exécution (10). On ne peut imposer, de la même façon, qu'un lot lié à une demande se termine avant la date de livraison prévue pour cette demande car le retard peut être autorisé dans l'ordonnancement. Par ailleurs, nous faisons l'hypothèse que les exécutions de deux lots d'une même recette requièrent une même ressource de capacité 1 et ne peuvent donc pas se chevaucher dans le temps. De ce fait, afin de casser la symétrie du problème, la date de fin d'un lot doit se trouver après la date de fin du lot précédent (11). En outre, ces dates doivent se situer dans la période considérée (12).

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \forall r' \in \mathcal{R}, \forall j' \in \{1, \ldots, max_{r'}\}, \forall m \in \mathcal{P}_r \cap \mathcal{C}_{r'},$$
$$t_{rj} \leq t_{r'j'} - \left(\alpha_{r'} + \lceil \beta_{r'} \times q_{r'j'} \rceil\right) + (End - Start) \times \left(1 - X^m_{rjr'j'}\right) \quad (10)$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r - 1\}, \quad t_{rj+1} \geq t_{rj} + (x_{rj+1} \times \alpha_r + \lceil \beta_r \times q_{rj+1} \rceil) \quad (11)$$

$$\forall r \in \mathcal{R}, \quad t_{r1} \geq Start + (x_{r1} \times \alpha_r + \lceil \beta_r \times q_{r1} \rceil) \quad (12a)$$
$$\forall r \in \mathcal{R}, \quad t_{rmax_r} \leq End \quad (12b)$$

3.2.4 Liaison des variables

Ce dernier ensemble de contraintes a pour but de relier les variables entre elles. Tout d'abord, pour casser la symétrie du problème, les contraintes (13), d'une part, expriment le fait que nécessairement, un nombre minimal de lots doit être exécuté, et (14), d'autre part, le fait que si le $j^{ème}$ lot est créé, c'est que le $j-1^{ème}$ l'est aussi. Par ailleurs, les contraintes (15) imposent que l'on respecte les décisions de planification. Les contraintes concernant le flux sur les arcs de pegging (16 à 20) forcent la quantité passant sur un arc à la valeur 0, si cet arc n'est pas créé et dans le cas contraire, bornent supérieurement cette quantité.

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, min_r\}, \quad x_{rj} = 1 \quad (13)$$
$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \quad x_{rj} \leq x_{rj-1} \quad (14)$$

$$\forall r \in \mathcal{R}, \quad Q_r = \sum_{j=1}^{max_r} q_{rj} \quad (15)$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \forall r' \in \mathcal{R}, \forall j' \in \{1, \ldots, max_{r'}\}, \forall m \in \mathcal{P}_r \cap \mathcal{C}_{r'}, \quad Q^m_{rjr'j'} \leq \min(\delta^m_r \times \overline{LS}_r, \delta^m_{r'} \times \overline{LS}_{r'}) \times X^m_{rjr'j'} \quad (16)$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\}, \forall m \in \mathcal{C}_r, \quad Q^m_{Irj} \leq \min(I^m_i, \delta^m_r \times \overline{LS}_r) \times X^m_{Irj} \quad (17)$$

$$\forall d \in \mathcal{D} \quad \backslash \quad d \text{ requiert } m, \quad Q_{Id} \leq \min(I^m_i, q_d) \times X_{Id} \quad (18)$$

$$\forall d \in \mathcal{D}, \forall r \in \mathcal{R} \quad \backslash \quad r \text{ produit m } d, \forall j \in \{1, \ldots, max_r\}, \quad Q_{rjd} \leq \min(q_d, \delta^m_r \times \overline{LS}_r) \times X_{rjd} \quad (19)$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\},$$
$$\forall m \in \mathcal{P}_r, \qquad Q^m_{rjI} \leq \min(I^m_f, \delta^m_r \times \overline{LS}_r) \times X^m_{rjI} \tag{20}$$

Les variables artificielles Z_{rjd}, $ZP^m_{rjr'j'}$ et $ZC^m_{rjr'j'}$ sont reliées respectivement aux t_d, t_{rj} et $t_{r'j'}$ d'une part et aux X_{rjd} et $X^m_{rjr'j'}$ d'autre part, par chacune trois équations, ce qui nous donne trois groupes d'équations (21, 22 et 23) :

$$\forall d \in \mathcal{D},$$
$$\forall r \in \mathcal{R} \quad \setminus \quad r \text{ produit m } d,$$
$$\forall j \in \{1, \ldots, max_r\}, \qquad Z_{rjd} \leq End \times X_{rjd} \tag{21a}$$
$$Z_{rjd} \geq t_{rj} - End \times (1 - X_{rjd}) \tag{21b}$$
$$Z_{rjd} \leq t_{rj} \tag{21c}$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\},$$
$$\forall r' \in \mathcal{R}, \forall j' \in \{1, \ldots, max_{r'}\},$$
$$\forall m \in \mathcal{P}_r \cap \mathcal{C}_{r'}, \qquad ZP^m_{rjr'j'} \leq End \times X_{rjr'j'} \tag{22a}$$
$$ZP^m_{rjr'j'} \geq t_{rj} - End \times (1 - X_{rjr'j'}) \tag{22b}$$
$$ZP^m_{rjr'j'} \leq t_{rj} \tag{22c}$$

$$\forall r \in \mathcal{R}, \forall j \in \{1, \ldots, max_r\},$$
$$\forall r' \in \mathcal{R}, \forall j' \in \{1, \ldots, max_{r'}\},$$
$$\forall m \in \mathcal{P}_r \cap \mathcal{C}_{r'}, \qquad ZC^m_{rjr'j'} \leq End \times X_{rjr'j'} \tag{23a}$$
$$ZC^m_{rjr'j'} \geq t_{r'j'} - End \times (1 - X_{rjr'j'}) \tag{23b}$$
$$ZC^m_{rjr'j'} \leq t_{r'j'} \tag{23c}$$

3.2.5 Fonction objectif

L'objectif peut alors s'écrire de la façon suivante :

$$\text{Minimiser} \quad \sum_{d \in \mathcal{D}} \sum_{\substack{r \in \mathcal{R} \setminus \\ \text{par } d, \in \mathcal{P}_r}} \sum_{j=1}^{max_r} (|X_{rjd} \times t_d - Z_{rjd}|)$$
$$+ \sum_{r \in \mathcal{R}} \sum_{j=1}^{max_r} \sum_{r' \in \mathcal{R}} \sum_{j'=1}^{max_{r'}} \sum_{m \in \mathcal{P}_r \cap \mathcal{C}_{r'}} (|ZC^m_{rjr'j'} - ZP^m_{rjr'j'}|)$$

4 Résultats numériques

4.1 Procédure de comparaison et instances résolues

La résolution du double problème de lot streaming et de pegging par ce programme mathématique ($\mathcal{P}_{lsp}$) est comparée à une résolution heuristique ($\mathcal{H}_{lsp}$) traitant séquentiellement le lot streaming (politique maximisant la taille des lots - donc minimisant leur nombre - et créant des lots de taille constante), puis le pegging (procédure gloutonne fondée sur un algorithme de tri).

Nous comparons différentes utilisations des méthodes (en ajoutant ou non les contraintes $\mathcal{C}o\mathcal{P}r$ lors de l'ordonnancement) et étudions la qualité des solutions obtenues pour l'ordonnancement final et le temps mis par la procédure de propagation de contraintes pour trouver cette solution. Cette étude a été réalisée sur un jeu de 54 instances qui se répartissent en 3 classes, en fonction de la présence ou non de modifications concernant le fonctionnement des ressources (fonctionnement continu ou non, productivité constante ou variable). Il s'agit de 7 chaînes de production comprenant un nombre de recettes compris entre 2 et 14, et un nombre de demandes entre 10 et 90, avec une unique étape de production (les recettes produisent directement les matériaux requis par les demandes). Chaque instance est obtenue en prenant en compte dans l'optimisation de la planification et de l'ordonnancement différents coûts (avance, retard, production) et en modifiant les coefficients pondérant ces coûts dans la fonction d'optimisation de chaque problème.

4.2 Tableaux de résultats et analyse

Nous regroupons dans un premier tableau (tab. 2) les résultats issus de la comparaison entre les valeurs des solutions pour l'ordonnancement et les temps de calcul pour chacune des exécutions possibles ($\mathcal{H}_{lsp}$ ou $\mathcal{P}_{lsp}$, sans ajout des contraintes $\mathcal{C}o\mathcal{P}r$). Le second tableau (tab. 3) présente l'impact sur le temps de la recherche de la présence de ces contraintes. Notons que nous ne prenons en considération, pour l'analyse des temps de calcul, que les instances pour lesquelles celui-ci est significatif (au minimum plusieurs secondes), c'est-à-dire 12 instances sur les 54. Nous employons les notations suivantes :

- $\mathcal{P}_{lsp} > \mathcal{H}_{lsp}$ quand $\mathcal{P}_{lsp}$ aboutit à une amélioration du critère considéré,
- $\mathcal{P}_{lsp} = \mathcal{H}_{lsp}$ quand $\mathcal{P}_{lsp}$ et $\mathcal{H}_{lsp}$ sont équivalentes,
- $\mathcal{P}_{lsp} < \mathcal{H}_{lsp}$ quand $\mathcal{P}_{lsp}$ aboutit à une dégradation du critère.

Analyse du tableau 2

Concernant la qualité des solutions de l'ordonnancement, on constate que dans 87% des cas $\mathcal{P}_{lsp}$ conduit à une solution meilleure (le plus souvent) ou équivalente à celle obtenue en utilisant $\mathcal{H}_{lsp}$. Comme notre ordonnancement est un problème de minimisation, l'amélioration d'une solution représente une diminution de la valeur de son objectif et la meilleure que l'on obtient est très élevée (elle correspond à une division par 3 de la solution issue de $\mathcal{H}_{lsp}$). De plus, lorsque $\mathcal{P}_{lsp}$ conduit à une moins bonne

Valeurs solutions	nombre d'instances	
	$\mathcal{P}_{lsp} > \mathcal{H}_{lsp}$	28/54 (51.8%)
	$\mathcal{P}_{lsp} = \mathcal{H}_{lsp}$	19/54 (35.2%)
	$\mathcal{P}_{lsp} < \mathcal{H}_{lsp}$	7/54 (13.0%)
	meilleure amélioration	67%
	plus mauvaise dégradation	9%
Temps calcul (12 instances significatives)	nombre d'instances	
	$\mathcal{P}_{lsp} > \mathcal{H}_{lsp}$	7/12 (58.3%)
	$\mathcal{P}_{lsp} < \mathcal{H}_{lsp}$	5/12 (41.7%)
	% meilleur gain	27%
	% plus mauvaise perte	67%

TAB. 2 – Comparaison $\mathcal{P}_{lsp}/\mathcal{H}_{lsp}$: solutions et temps de calcul

$\mathcal{H}_{lsp}$	gain en temps	11/12 (91.7%)
	meilleur gain	7.5%
	plus mauvaise perte	1%
$\mathcal{P}_{lsp}$	gain en temps	9/12 (75.0%)
	% meilleur gain	41%
	% plus mauvaise perte	63%

TAB. 3 – Gain en temps de calcul quand les contraintes $\mathcal{C}o\mathcal{P}r$ ne sont pas ajoutées

solution, le dégradation est faible, on reste donc dans le même ordre de grandeur pour l'objectif de la solution d'ordonnancement.

L'analyse des temps de calcul n'est faite que sur 22% des instances en raison des faibles valeurs mesurées, cependant on peut tout de même extraire quelque conclusion. L'utilisation de $\mathcal{P}_{lsp}$ conduit à une diminution du temps de recherche de l'ordonnancement pour 7 instances sur les 12 significatives avec un gain maximal relativement élevé, néanmoins, lorsque le temps de recherche est augmenté, il peut l'être de beaucoup.

Analyse du tableau 3

Nous l'avons dit en section 2.2.2, l'un des intérêts du pegging consiste à éviter d'avoir recours à l'introduction des contraintes $\mathcal{C}o\mathcal{P}r$ dont la propagation constitue parfois le goulet d'étranglement de la procédure de recherche de l'ordonnancement. Nous comparons donc les temps de calcul pour les mêmes instances de problèmes dans lesquelles on introduit ou non ces contraintes. A nouveau cette étude n'est réalisée que sur les 12 instances pour lesquelles on a relevé des mesures significatives. Dans le cas où l'on utilise l'heuristique $\mathcal{H}_{lsp}$, on observe un gain de temps sur toutes les instances, à l'exception d'une seule, cependant la perte observée dans ce cas est négligeable (1%). L'utilisation d'un pegging issu de $\mathcal{P}_{lsp}$ ne montre pas de manière aussi évidente l'intérêt de se dispenser des contraintes $\mathcal{C}o\mathcal{P}r$, a priori gênantes, 9 instances sur 12 aboutissent tout de même à un gain de temps et, contrairement à $\mathcal{H}_{lsp}$, ce gain peut être très élevé, toutefois,

on peut observer une perte de temps encore plus élevée.

5 Conclusion et perspectives

La problématique de l'intégration de la planification de la production et de l'ordonnancement est cruciale pour les entreprises appartenant à l'industrie de transformation. L'architecture que nous avons étudiée pour ce problème se découpe en trois étapes : planification, lot streaming/pegging et ordonnancement, la seconde est celle à laquelle nous nous sommes intéressés ici. Elle consiste en le dimensionnement des lots de production ordonnançables ainsi qu'en l'établissement précis des flux de matériaux entre les entités de la chaîne de production.

Un modèle de programmation linéaire en variables mixtes a été présenté pour résoudre ce problème. La résolution de 54 instances par ce modèle a été comparée à une procédure naïve séquentielle. Les résultats numériques laissent à penser qu'une telle approche peut effectivement améliorer la qualité de l'ordonnancement final, ainsi qu'accélérer le temps de calcul de la procédure de recherche de celui-ci. Un modèle dans lequel le problème serait divisé en deux sous-problèmes, résolus séparément mais de manière coopérative est l'une des orientations envisagées pour la suite de notre étude de cette problématique.

Références

BAKER K. & JIA D. (1993). A comparative study of lot streaming procedures. *OMEGA International Journal of Management Science*, **21**(5), 561–566.

DAUZÈRE-PÉRÈS S. & LASSERRE J. B. (1994). *An Integrated Approach in Production Planning and Scheduling*. Springer.

HOPP W. & SPEARMAN M. (2000). *Factory Physics*. Mc Graw-Hill/Irwin.

LUNN T. & NEFF S. A. (1992). *MRP, Integrating Material Requirements Planning and Modern Business*. Irwin.

MARAVELIAS C. & GROSSMANN I. (2005). An hybrid milp/cp decomposition approach for the continuous time scheduling of multipurpose batch plants. *Computers and Chemical Engineering*, **28**(10), 1921–1949.

ROUX W. (1997). *Une approche cohérente pour la planification et l'ordonnancement de systèmes de production complexes*. PhD thesis, Université Paul Sabatier, Toulouse.

STAGGEMEIER A. & CLARK A. (2001). A survey of lot-sizing and scheduling models. In *23rd Symposium of the Brazilian Operational Research Society (SOBRAPO)*.

TIMPE C. (2002). Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum*, **24**(4).

TRIETSCH D. & BAKER K. (1993). Basic techniques for lot streaming. *Operation Research*, **41**(6), 1065–1076.

Algorithmes d'Itération de la Politique Symboliques et Heuristiques pour les Problèmes de Planification d'Exploration Structurés

Florent Teichteil-Königsbuch[1], Patrick Fabiani[1]

Office National d'Etudes et de Recherches Aérospatiales
Département Commande des Systèmes et Dynamique du vol
2, Avenue Edouard Belin
31055 Toulouse Cedex 4
florent.teichteil@cert.fr et patrick.fabiani@cert.fr

Résumé :

Le projet ReSSAC, un projet d'aéronef d'exploration autonome à l'ONERA, motive notre recherche sur la planification de décision en environnement incertain dans des espaces d'état de grande taille, à l'aide de Processus Décisionnels de Markov. Les problèmes de planification d'exploration comprennent plusieurs buts intermédiaires et sont structurés en deux composantes. Un graphe des états énumérés représente la composante de navigation du problème. Un ensemble de variables d'état décrit, d'une façon compacte et implicite, les autres caractéristiques du problème, dont les buts intermédiaires qui doivent être atteints en séquence. Nous comparons des algorithmes de Programmation Dynamique avec Recherche Heuristique dans des PDM de grande taille sur la base de tels problèmes d'exploration. Un schéma algorithmique commun est utilisé pour comparer LAO* avec un algorithme d'itération de la politique sous-optimal appelé Programmation Dynamique Symbolique Focalisée (SFDP), tel que le temps d'optimisation et la qualité de la solution sont contrôlés en planifiant sur un sous-ensemble sélectionné de buts intermédiaires. Nous présentons un algorithme encore plus rapide, sfDP, qui trouve rapidement des solutions à des problèmes très grands. La version incrémentale IsfDP appelle itérativement sfDP sur une liste croissante de buts intermédiaires.

Mots-clés : planification, programmation dynamique stochastique, processus décisionnels markoviens, robotique autonome

1 Introduction

Les Processus Décisionnels de Markov (PDM) (Puterman, 1994) constituent un cadre de référence pour la décision séquentielle sous incertitude : afin de prendre en compte

les effets incertains des actions de l'agent, une *politique* est calculée sur l'espace d'états. C'est une fonction qui fait correspondre à chaque état énuméré possible, l'action qui doit être effectuée ensuite. La valeur moyenne des récompenses accumulées en suivant cette politique est la *fonction de valeur*. La *politique optimale* et la *fonction de valeur optimale* correspondante maximisent la probabilité de succès, ou l'espérance mathématique des récompenses futures. Les algorithmes stochastiques classiques de programmation dynamique sont basés sur un espace d'états explicitement énuméré et non structuré, dont la taille est une fonction exponentielle du nombre de caractéristiques qui décrivent le problème. L'énumération des états elle-même peut devenir rapidement insurmontable pour des problèmes réalistes. Plus généralement, (Boutilier *et al.*, 1999) fournissent une discussion exhaustive sur les problèmes de complexité et de modélisation. (Boutilier *et al.*, 2000) montrent les bénéfices des représentations factorisées dans le but d'éviter l'énumération des états, de raisonner à un niveau d'abstraction plus élevé comme dans (Dearden & Boutilier, 1997), et de prendre en compte des aspects non markoviens du problème, comme les récompenses ou les buts qui dépendent de l'historique des états (voir (Bacchus *et al.*, 1997)). D'autres approches, comme dans (Dean & Lin, 1995), introduisent une décomposition hiérarchique de l'espace d'états en sous-régions. Des politiques locales sont ensuite calculées dans chaque sous-région et deviennent les macro-actions applicables dans les macro-états d'un PDM global abstrait et factorisé. (Teichteil-Königsbuch & Fabiani, 2004) proposent de combiner à la fois des techniques de décomposition et de factorisation. D'autres contributions importantes, telles que la librairie SPUDD (Hoey *et al.*, 2000), ont amélioré l'efficacité des algorithmes de résolution des PDM factorisés en utilisant des diagrammes de décision, tirés de la communauté de Vérification de Modèle. Récemment, des procédés de recherche heuristique ont été proposés tels *LAO** de (Hansen & Zilberstein, 2001), *LRTDP* de (Bonet & Geffner, 2003a) ou *sRTDP* de (Feng & Hansen, 2002).

2 Itération de la politique symbolique heuristique

L'algorithme de recherche heuristique *Symbolic Focused Dynamic Programming (SFDP)* est une contribution originale de notre part. Comme *LAO**, *SFDP* procède en deux phases : l'*expansion de l'espace de planification* est suivie d'une *optimisation*. L'*expansion de l'espace de planification* est basée sur une étude d'atteignabilité qui utilise la politique courante et les *buts de planification*. L'étape d'*optimisation* est une phase de programmation dynamique appliquée sur l'espace de planification précédemment atteint par la phase d'expansion : ceci permet de *focaliser plus ou moins* la recherche et donc de contrôler le processus d'optimisation. Ainsi, une première solution est rapidement trouvée qui peut être améliorée plus tard si le temps est disponible.

Dans la suite, nous présentons d'abord nos motivations pour la planification de décision symbolique et heuristique sur la base d'une famille de problèmes d'exploration de type «grille». Notre approche exploite la structure du problème en termes de décomposition et de factorisation (voir aussi (Teichteil-Königsbuch & Fabiani, 2005)) : les problèmes de planification d'exploration motivent naturellement la décomposition de l'espace de navigation d'une part, et l'utilisation d'un modèle factorisé en variables de mission d'autre part. Nous proposons un procédé heuristique commun pour la réso-

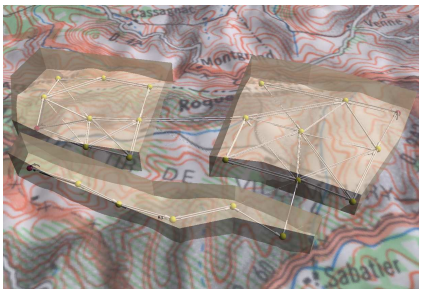
lution des problèmes de planification d'exploration. Notre travail est inspiré des algorithmes d'itération de la valeur symbolique *sLAO** (Feng & Hansen, 2002) et *sRTDP* (Feng *et al.*, 2003). Nous décrivons nos implémentations et comparaisons des algorithmes *SFDP* d'*itération de la politique* et de la *valeur*. L'intérêt de *SFDP*, qui focalise la recherche de la solution optimale sur des buts de planification à réaliser, est présenté à travers des comparaisons de performance par rapport à *sLAO** et *SPUDD* : *SFDP* trouve plus rapidement que *sLAO** et *SPUDD* des solutions sous-optimales, dont la qualité peut être contrôlée par le nombre de buts de planification imposés à la solution. Nous avons également implémenté une version d'*itération de la politique* de *SPUDD* ainsi que des versions symboliques d'*itération de la valeur* et de la *politique* de *LAO**. Une version de *SFDP* sous-optimale mais plus rapide, *sfDP*, est présentée. Une version incrémentale de *sfDP*, nommée *IsfDP*, appelle itérativement *sfDP* afin d'améliorer incrémentalement la solution. Dans ce papier, nous ne présentons pas de version optimale de *SFDP*. Des expérimentations ont été effectuées sur une même famille de problèmes d'exploration de type «grille». Nous concluons en invoquant des travaux futurs sur les algorithmes de Programmation Dynamique avec Recherche Heuristique.

3 Motivations

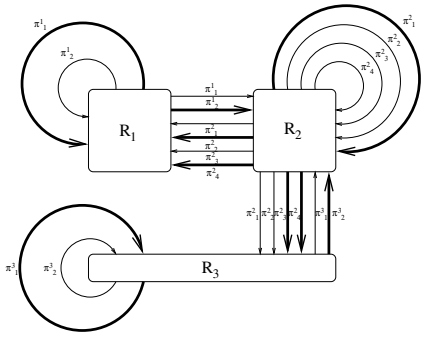
Notre recherche est motivée par une application d'exploration pour un aéronef autonome de « recherche et sauvetage». Nous souhaitons contrôler le processus d'optimisation, la qualité de la solution et le temps d'optimisation, sous la contrainte d'atteindre des buts spécifiques avec une priorité maximale. Une solution doit être rapidement disponible en cas de replanification, pour être améliorée plus tard vers la solution optimale si le temps est disponible. D'autre part, certains sous-buts spécifiques du problème peuvent être imposés dans la solution indépendamment des autres sous-buts. Une mission d'exploration comprend un problème de navigation (voir figure 1.a) dans un environnement partiellement connu d'une part, et un problème d'acquisition d'informations en ligne et de replanification d'autre part. Des buts alternatifs ou intermédiaires peuvent être imposés à l'agent, soit en tant que choix alternatifs (l'agent peut atteindre son but final O_f en réalisant préalablement O_1 ou O_2) soit en tant que réalisations successives (l'agent doit atteindre O_f en réalisant O_1 puis O_2). Certains buts finaux, tel que l'atterrissage dans une zone sûre, doivent être atteints dans tous les plans possibles. Certains buts, comme l'exploration ou la recherche d'une région, sont les préconditions devant être vérifiées avant de chercher à réaliser des buts supplémentaires. Certaines récompenses et certains buts de l'agent dépendent de l'historique des états, et il peut exister des contraintes d'ordre entre les buts, i.e. l'agent doit prendre de l'information sur la région R_k et la transmettre à son centre de contrôle au sol avant de procéder à une phase de navigation vers les régions voisines. D'autres variables d'état, comme le niveau d'autonomie en énergie A de l'agent, sont clairement orthogonales aux composantes de navigation, mais elles n'en sont pas du tout découplées : chaque déplacement de l'aéronef consomme de l'énergie, qui est modélisé comme une probabilité de transition vers une valeur inférieure de la variable d'autonomie en énergie. Un niveau insuffisant d'autonomie en énergie peut forcer l'agent à *Abandonner* sa mission et à retourner à sa base, ou à atterrir sur une base d'urgence ou de crash sécurisée. Enfin,

il peut être spécifié, comme dans notre exemple simple, que les récompenses ne peuvent être obtenues qu'une seule fois, si bien qu'avoir atteint le but O_1 dans la région R_1 annule la récompense correspondante et change ainsi complètement la stratégie optimale dans cette région. Calculer une stratégie pour chaque combinaison possible de buts O_j en cours de réalisation ou non, mène à effectuer un nombre de résolutions qui est exponentiel en le nombre de tels buts intermédiaires possibles. Des contraintes d'ordre peuvent aussi être imposées sur la séquence de réalisation de certains sous-buts.

Un tel problème est non markovien. (Bacchus *et al.*, 1997) utilisent une logique temporelle pour décrire et raisonner sur de telles contraintes et générer les «variables additionnelles» nécessaires afin de prendre en compte l'aspect non markovien du problème. Dans l'état actuel de nos travaux, nous introduisons ces variables d'état additionnelles «à la main» dans le problème représenté sur la figure 1.a. Nous encodons les contraintes ci-dessus sous forme de *Réseaux Bayésiens Dynamiques (RBD)* (Dean & Kanazawa, 1989) comme dans la figure 2. Dans tout ce papier, nous utilisons une famille de problèmes similaires de tailles et de complexités différentes, mais tous de type «grille», avec des régions faiblement couplées et des variables d'état additionnelles, en particulier des variables qui décrivent quels buts ont été atteints ou non. Les grilles sont utilisées car elles permettent de générer facilement des problèmes de grande taille comme dans (Bonet & Geffner, 2003b). Nous utilisons des problèmes simples afin d'expliquer notre approche, et d'autres plus compliqués pour démontrer l'efficacité de l'approche.



(a) Environnement initial



(b) PDM abstrait décomposé en 3 régions

FIG. 1 – Composante de navigation d'un problème de type « grille» : 3 régions et 3 récompenses qui ne peuvent être obtenues qu'une seule fois et à tour de rôle

4 Décomposition de notre exemple simple

La décomposition du problème est motivée par des considérations d'optimisation du temps de calcul. Les travaux de (Teichteil-Königsbuch & Fabiani, 2005) proposent de factoriser de tels problèmes par une décomposition hiérarchique en un PDM abstrait factorisé, dont les bénéfices en terme d'efficacité sont montrés. (Hauskrecht *et al.*, 1998) ou (Parr, 1998) ont proposé deux algorithmes pour le calcul des politiques locales durant la phase de décomposition du PDM. Selon (Teichteil-Königsbuch & Fabiani, 2005),

cette dernière approche (*PL* par (Parr, 1998)) est celle qui offre les meilleures performances et flexibilité. Afin de prendre en compte le fait que les récompenses ne peuvent être obtenues qu'une seule fois, nous devons adapter l'algorithme de (Parr, 1998) à notre problème. L'optimisation des politiques locales des régions doit être conditionnée par le statut des buts de la région : en pratique, ceci limite grandement le nombre de cas puisque la difficulté combinatoire est répartie dans les régions. Dans notre exemple simple, nous avons seulement un but par région, ce qui mène à optimiser 2 ensembles de *politiques locales conditionnelles* par région : un si le *but* local n'a pas encore été atteint par l'agent, et un autre s'il l'a atteint.

Le bénéfice direct de la décomposition vient du fait que si le problème comprend k régions et un seul but par région, seulement $2k$ ensembles de politiques locales sont calculés. Sans la décomposition, 2^k politiques locales devraient être optimisées. Suite à la phase de décomposition, la composante de navigation (sous forme de graphe) du problème d'exploration est divisée en *macro-états*, chacun correspondant à une région (voir figure 1.b), et combinée aux autres variables d'état orthogonales. Le PDM abstrait résultant est sous une forme factorisée et peut être représenté par des *Réseaux Bayésiens Dynamiques (RBD)* comme dans la figure 2. La variable d'état R représente la région, O_1, O_2 et O_3 représentent les *buts*, et A le niveau d'autonomie en énergie de l'agent. Pour des raisons de simplicité, nous supposons que le niveau d'autonomie en énergie est binaire avec une consommation constante parmi les régions : la fonction f indiquant la probabilité de «perdre le niveau minimal d'autonomie en énergie» entre deux instants est $f(R_i, R_j, \pi) = [0,9\ ;\ 0,1]$ pour tous i, j et π.

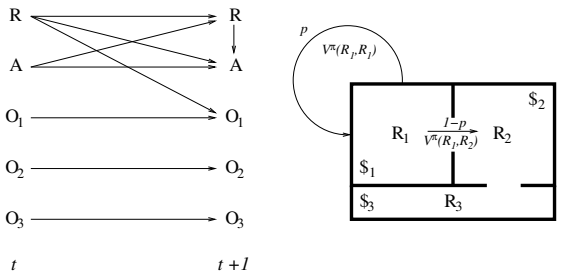


FIG. 2 – Réseau d'action du PDM abstrait factorisé et transitions du PDM abstrait pour les politiques locales de la région R_1

5 Diagrammes de Décision Algébriques

Dans les RBDs, les probabilités et les récompenses de transition sont représentées par des *Tables de Probabilités Conditionnelles*, qui peuvent être implémentées sous forme de «diagrammes de décision algébriques» (DDA) comme dans (Hoey *et al.*, 2000). L'intérêt des DDAs réside dans la fusion des nœuds de même valeur, menant à un graphe au lieu d'un arbre (voir figure 4) et dans la définition d'une valeur non stockée en mémoire et codant une valeur fréquemment utilisée, comme dans les matrices creuses (typiquement 0 pour les PDMs). La figure 3 montre des exemples d'arbres de probabilité de

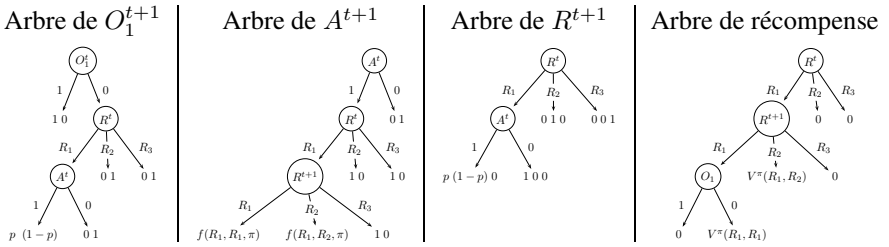


FIG. 3 – Arbres de probabilité de O_1^{t+1}, A^{t+1} et R^{t+1} et arbre de récompense des réseaux d'action correspondants aux politiques locales de la région R_1

transition (les feuilles sont des vecteurs) et d'arbre de récompense de transition pour les macro-actions de la région R_1 de notre exemple de PDM abstrait. Pour chaque variable d'état post-action, chaque feuille de l'arbre de probabilité est une liste contenant les probabilités d'obtenir chaque valeur possible x_i^{t+1} de cette variable, connaissant les valeurs des autres variables x_i^t, x_j^t, x_j^{t+1} le long du chemin $x_i^t \wedge \left(\wedge_{j\neq i}\left(x_j^t \wedge x_j^{t+1}\right)\right)$ menant de la racine de l'arbre à la feuille considérée. La figure 3 exprime le fait que, pour obtenir la récompense associée à un but donné, ce but ne doit pas être encore atteint et l'agent doit d'abord atteindre la région correspondante avec un niveau d'autonomie en énergie suffisant. Les buts des autres régions ne peuvent pas être atteints depuis l'«intérieur» si bien que les arbres de décision correspondants se résument à ces variables *buts* dont l'état reste inchangé. D'après (Hoey *et al.*, 2000), la plage de valeurs des variables d'état a une influence importante sur le temps de calcul. C'est donc une raison supplémentaire pour décomposer la composante de navigation dans notre problème d'exploration en peu de régions abstraites agrégées, car les variables de position géographique ou de navigation ont typiquement un grand nombre de valeurs possibles.

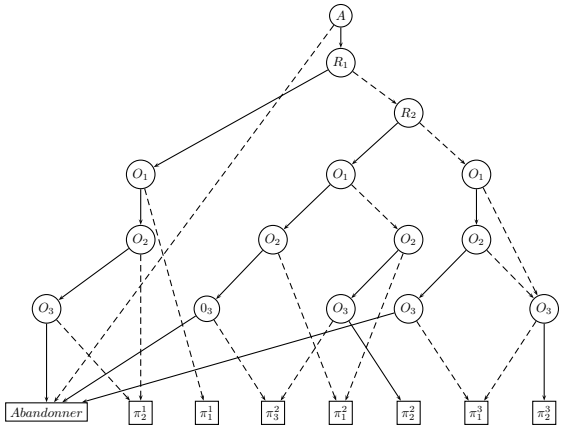


FIG. 4 – DDA de politique optimal du PDM abstrait factorisé de la figure 2

L'implémentation de nos versions d'*itération de la politique*, basées sur *SPUDD* de (Hoey *et al.*, 2000), a nécessité quelques développements techniques : la politique cou-

rante est représentée par une liste de *Diagrammes de Décision Binaires (DDB)*, plus efficaces que les DDA, chaque DDB encodant le domaine d'application d'une action pour cette politique. De plus, afin d'améliorer l'efficacité de nos algorithmes, nous appliquons pour chaque action un masque sous forme de DDB représentant le domaine d'application de l'action indépendamment de la politique courante. Enfin, pour notre algorithme *SFDP* décrit dans la suite, les DDB et les DDA sont restreints à l'espace d'état atteignable courant qui est représenté par un masque encodé sous forme de DDB.

6 Programmation Dynamique Symbolique avec Recherche Heuristique

Fonction PDSRH($\mathcal{E}, \mathcal{A}, \mathcal{I}, \mathcal{B}$: **DDB** $\mathcal{T}$,$\mathcal{R}$: **DDA**) : $\mathcal{E}_k, \Pi_k, V_k$
  $E, \mathcal{E}_k$: **DDB** V_k : **DDA** Π_k : **liste<DDB>** $E \longleftarrow Atteignable(\mathcal{I}, \mathcal{A}, \mathcal{B})$
  $(\Pi_0, P(\mathcal{I})) \longleftarrow PlusCourtCheminStochastique(E, \mathcal{I}, \mathcal{B}, \mathcal{T}_{|E})$
  $\mathcal{E}_0 \longleftarrow FiltreEtats(E, P(\mathcal{I}), \epsilon)$
  $k \longleftarrow 0$
  Répéter
    Selon que
      $Algorithme = SFDP : \mathcal{E}_{k+1} \longleftarrow Atteignable(\mathcal{I}, \Pi_k, \mathcal{B})$
      $Algorithme = sLAO^* : \mathcal{E}_{k+1} \longleftarrow Atteignable(\mathcal{E}_k, \Pi_k,$ 1 étape en avant$)$
    Fin Selon que
    $(V_{k+1}, \Pi_{k+1}) \longleftarrow ProgrammationDynamique(\mathcal{E}_{k+1}, \mathcal{T}_{|\mathcal{E}_{k+1}}, \mathcal{R}_{|\mathcal{E}_{k+1}})$
    $k \longleftarrow k + 1$
  jusqu'à ce que $(\|V_{k+1} - V_k\| < \epsilon$ sur $\mathcal{E}_k)$
  Retourner $(\mathcal{E}_k, V_k, \Pi_k)$
Fin

Algorithme 1: Schéma général de la Programmation Dynamique Symbolique avec Recherche Heuristique (fonction PDSRH)

L'algorithme *sLAO** (Feng & Hansen, 2002) est la version symbolique de *LAO** (Hansen & Zilberstein, 2001). Afin de comparer avec notre algorithme *SFDP* décrit dans la suite, nous avons implémenté l'algorithme *sLAO** en version Itération de la Politique, représenté dans l'algorithme 1 (notations détaillées plus loin) pour $Algorithme = sLAO^*$, où la fonction générique $Atteignable$ est appliquée depuis S_k à chaque itération, en utilisant les actions de la politique courante π_k et en faisant *1 étape en avant*. Notons que S_k dans *sLAO** est supposé grandir en permanence, ce qui dans ce contexte donne à *sLAO** la même garantie d'optimalité que *LAO** (Hansen & Zilberstein, 2001) : lorsque l'algorithme converge, chaque trajectoire suivie dans S_k en appliquant la politique courante termine dans un état but. Comme *sRTDP* (Feng *et al.*, 2003), notre algorithme *SFDP* applique la fonction $Atteignable$ depuis les états initiaux I à chaque itération, en utilisant les actions de la politique courante π_k. Néanmoins, à la différence de *sRTDP* qui est basé sur l'étude de trajectoires tirées aléatoirement au cours de plusieurs essais, *SFDP* calcule l'espace d'états atteignables et la politique optimale en raisonnant globalement sur tout l'espace d'états atteignables courant. Les algorithmes de Programmation Dynamique Symbolique avec Recherche Heuristique suivent un pro-

cédé commun en deux phases, représenté dans l'algorithme 1 :
- une première phase d'**étude d'atteignabilité** et de calcul **heuristique**,
- une seconde phase consécutive de **programmation dynamique**.

Ce procédé constitue notre base algorithmique commune pour le développement et la comparaison de différents algorithmes de programmation dynamique avec recherche heuristique, conformes à *sLAO** de (Feng & Hansen, 2002) et *sRTDP* de (Feng *et al.*, 2003). La figure 5.b illustre une comparaison schématique entre les algorithmes *sLAO** et *sRTDP* décrits dans la littérature, et nos algorithmes *sfDP* et *IsfDP* dérivés de *SFDP*, que nous présentons dans la suite.

6.1 Etude d'atteignabilité

Il est important de remarquer que l'application d'un algorithme de programmation dynamique sur l'espace d'états complet est impossible pour des problèmes réalistes comprenant beaucoup de variables d'état, comme nos problèmes d'exploration. L'espace de travail est donc calculé avec précaution à chaque itération, en le gardant petit mais en balayant quand même l'espace d'états complet. Des calculs **heuristiques**, telle l'*analyse de plus court chemin stochastique* proposée interviennent durant cette première phase, essentiellement pour fournir une estimation *admissible* de la fonction de valeur ou, de manière équivalente, une politique initiale sur la bordure de l'espace de planification (voir figure 5.a). La phase d'«expansion de l'espace de planification» permet de contrôler le processus d'optimisation. *sLAO** augmente incrémentalement son espace de travail jusqu'à ce que toutes les récompenses pertinentes aient été collectées et que le processus converge.

6.1.1 Analyse d'atteignabilité déterministe

Nous appelons $Atteignable(\mathbb{E}, \Pi_A, Stop)$ une fonction qui prend en entrée un ensemble d'états $\mathbb{E}$, un ensemble d'actions applicables $\Pi_A \subset \mathcal{A}$ où $\mathcal{A}$ est l'ensemble des transitions **déterministes**, et une condition d'arrêt $Stop$. Cette fonction calcule l'ensemble E de tous les états atteignables depuis $\mathbb{E}$ avec des applications successives des actions déterministes de Π_A dans une boucle itérative qui s'arrête lorsque la condition $Stop$ est vérifiée : i.e. $Stop$ peut être $\mathcal{B} \subset E$ ou *1 étape en avant*. Les actions sont rendues déterministes en mettant les transitions de probabilité non nulle à 1, ce qui nous permet de convertir les DDA en DDB qui sont plus efficaces. L'idée sous-jacente est que l'expansion de l'espace de planification est contrôlé via la condition $Stop$, liée à la réalisation de buts de planification spécifiques.

6.1.2 Analyse de plus court chemin stochastique

Dans cette étape, nous calculons une politique (ou fonction de valeur) heuristique initiale et réduisons – si possible – l'espace d'états atteignables initial. Nous appelons $PlusCourtCheminStochastique(E, \mathcal{I}, \mathcal{B}, \mathcal{T}_{|E})$ une fonction qui prend en entrée l'espace des états déterministes atteignables E, l'espace des buts à atteindre $\mathcal{B}$ et l'ensemble des transitions stochastiques $\mathcal{T}_{|E}$ pour toutes les actions applicables dans E. Elle calcule le plus court chemin stochastique depuis n'importe quel état de E vers

un état de $\mathcal{B}$ quelconque, en utilisant les transitions stochastiques de $\mathcal{T}_{|E}$ sans les récompenses associées. Elle renvoie la politique de plus court chemin stochastique Π_0 ainsi que la probabilité $P(\mathcal{I})$ d'atteindre $\mathcal{B}$ depuis $\mathcal{I}$ en appliquant Π_0. Nous appelons $FiltreEtats(E, P(\mathcal{I}), \epsilon)$, une fonction qui filtre les états de E qui ont une faible probabilité $P(e)$ d'atteindre l'ensemble des buts $\mathcal{B}$ comparés à ceux de $\mathcal{I}$, dont la probabilité d'atteindre $\mathcal{B}$ est $P(\mathcal{I}) : P(e) < \epsilon \cdot P(\mathcal{I})$. Au niveau des résultats, le filtrage stochastique des états atteignables semble être très comparable à l'effet du tirage aléatoire dans *sRTDP* de (Feng *et al.*, 2003), puisque les transitions de faibles probabilités ont peu de chances d'être tirées, au point que les états atteignables avec de telles transitions ont peu de chances d'être visités.

7 SFDP et sLAO*

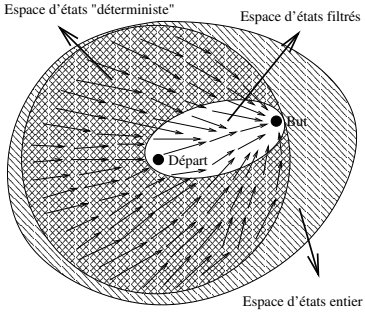
Durant cette étape, la solution du PDM est optimisée sur l'espace d'états atteignables courant. L'expansion de l'espace d'états courant résulte de la fonction $Atteignable$, appliquée soit depuis $\mathcal{I}$ (*SFDP*), soit depuis l'espace d'états atteignables précédent (*sLAO**). La fonction $Atteignable$ étend l'ensemble des états atteignables jusqu'à ce que **tous les états qui satisfont les conditions des buts** soient atteints. La fonction $ProgrammationDynamique(\mathbb{E}, \mathcal{T}_{|\mathbb{E}}, \mathcal{R}_{|\mathbb{E}})$ de l'algorithme principal 1 peut être $IterationValeur$ ou $IterationPolitique$: elle calcule une politique optimale sur un ensemble $\mathbb{E}$ connaissant les transitions stochastiques $\mathcal{T}_{|\mathbb{E}}$ et les récompenses associées $\mathcal{R}_{|\mathbb{E}}$ restreintes à $\mathbb{E}$.

Pour *SFDP*, la fonction générique $Atteignable$ est appliquée depuis l'ensemble des états initiaux possibles $\mathcal{I}$ à chaque itération, en utilisant les actions de la politique courante Π_k jusqu'à ce que $\mathcal{B}$ soit atteint (voir l'algorithme 1 avec $Algorithme = SFDP$). De fait, il n'est pas du tout garanti que l'espace de travail $\mathcal{E}_k$ grandisse : au contraire, *SFDP* a été conçu de manière à focaliser la recherche vers les parties cohérentes de l'espace d'états (voir figure 5.a). Par conséquent, *SFDP* ne fournira pas la solution optimale du problème, mais plutôt la «solution la plus rapide», à moins que *SFDP* soit contraint à visiter toutes les récompenses de l'espace d'états lorsque toutes les récompenses du problème ont été données comme *contraintes de buts de planification* préalablement à l'optimisation.

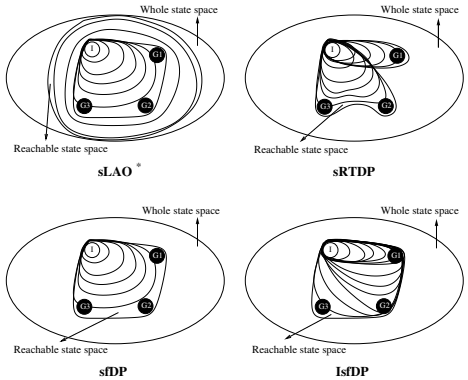
Pour *sLAO**, la fonction générique $Atteignable$ est appliquée depuis $\mathcal{E}_k$ à chaque itération, en utilisant les actions de la politique courante Π_k, avec 1 étape en avant (voir l'algorithme 1 avec $Algorithme = sLAO^*$). Contrairement à *SFDP*, $\mathcal{E}_k$ dans *sLAO** est supposé toujours grandir, ce qui lui garantit dans ce contexte d'être optimal.

8 Expérimentations

Nous avons conduit nos expérimentations sur des problèmes de type «grille» inspirés de l'exemple de la figure 1.a et dont le modèle abstrait factorisé est défini par les arbres de transition de la figure 3 pour chaque région. Le nombre de nœuds du graphe de navigation est $45 \times 45 = 2025$ et le nombre total d'états augmente exponentiellement avec le nombre de buts et de régions du problème ($+1$ pour le niveau d'énergie). Les



(a) L'heuristique de *SFDP* focalise la recherche des buts de planification.



(b) Espaces d'états atteignables de *sLAO**, *sRTDP*, *sfDP* et *IsfDP*

FIG. 5 – Principes de l'étude d'atteignabilité avec recherche heuristique

problèmes linéaires, également testés, sont des PDM de type «grille» où chaque région est reliée seulement à une région précédente et une région suivante.

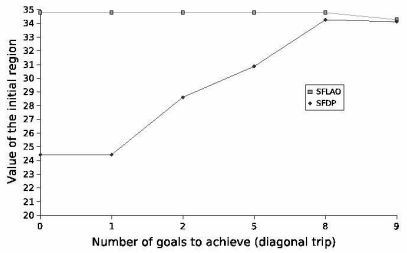
Nous avons d'abord effectué la comparaison entre les approches à base de PDM abstrait factorisé et de PDM énuméré. Les résultats sont présentés dans le tableau 1 qui montre que la modélisation de l'espace d'états est réellement un problème crucial. Le temps nécessaire pour construire les structures de données des transitions pour le PDM énuméré illustre l'handicap de l'approche énumérée.

type	taille	régions	modèle	construction	décomposition	macro-actions	résolution
linéaire	384	3	factorisé	$< 0{,}01$	0,08	10	0,02
			énuméré	0,03	–	–	0,01
	$6 \cdot 10^3$	6	factorisé	0,01	0,38	33	1,51
			énuméré	4,21	–	–	0,38
	$7 \cdot 10^4$	9	factorisé	0,03	0,56	47	26,8
			énuméré	587,62	–	–	5,03
concentrique	$8 \cdot 10^5$	9 (9 e./r.)	factorisé	0,02	0,13	21	0,12
			énuméré	746,98	–	–	2,25
	$7 \cdot 10^6$	9 (81 e./r.)	factorisé	0,02	40,61	61	16,77
			énuméré	> 1hr	–	–	–

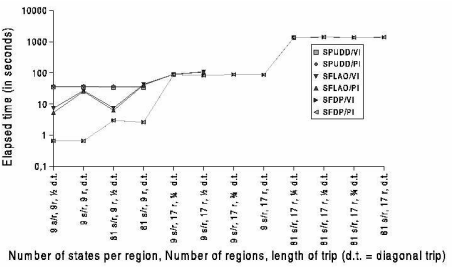
TAB. 1 – Comparaison entre les approches à base de PDM abstrait factorisé – résolu avec *SPUDD* – et de PDM énuméré (temps de calcul en secondes)

Nous présentons ensuite une comparaison de six algorithmes, dont les deux derniers font partie de notre contribution, que nous avons implémenté sur la base de l'algorithme d'*itération de la valeur* SPUDD/VI : 1.SPUDD/VI – 2.SPUDD/PI – 3.sLAO/VI – 4.sLAO/PI – 5.SFDP/VI – 6.SFDP/PI. Nous avons comparé la qualité des politiques solutions de *SFDP* lorsqu'un nombre croissant de contraintes est imposé concernant les buts de planification (voir figure 6.a). Il apparaît que *SFDP* est bien plus sensible aux contraintes de buts que *sLAO**. Imposer à *SFDP* de réaliser TOUS les buts mène l'algorithme à se comporter comme *sLAO**, en étendant continuellement son espace

de planification sans focalisation, tant que le problème correspondant peut être résolu. Au contraire, *sLAO** tend à essayer et à atteindre toutes les récompenses et les buts du problème même si ce n'est pas demandé. Le temps de calcul correspondant augmente en proportion du nombre de combinaisons d'alternatives possibles.



(a) Qualité de la solution avec un nombre croissant de buts de planification à réaliser (9 états/région, 9 régions)



(b) Temps de résolution pour des problèmes de complexité croissante

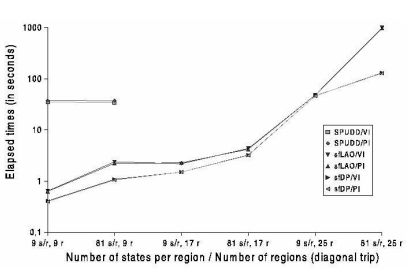
FIG. 6 – Comparaison entre *SFDP* et *sLAO**

De manière similaire, nous avons comparé les temps de calcul pour *sLAO** et *SFDP* sur des problèmes dont la taille de l'espace d'états augmente soit en rajoutant des régions, soit en rajoutant des états géographiques dans chaque région et en faisant varier les points initiaux et buts (voir figure 6.b). La conclusion est que *SFDP* trouve assez rapidement des solutions (sous-optimales) pour les problèmes les plus complexes que nous avons pu décomposer en temps raisonnable (248 s.) et sans que la mémoire vive ne swappe, ce qui correspond à un déplacement diagonal. Au contraire, *sLAO** ne peut donner aucune réponse après plus d'une heure de calcul dès le quatrième problème. Une telle comparaison doit être analysée avec précaution : *SFDP* ne peut pas être considéré comme «meilleur» ou «préférable» sur la base de cette comparaison. D'autre part, la figure 6.a montre qu'il est possible d'établir un profil de la qualité de la réponse de *SFDP* sur certaines classes de problèmes. La réponse rapide donnée par cet algorithme pourrait être réutilisée afin d'initialiser une politique, ou une fonction de valeur, heuristique admissible pour d'autres algorithmes.

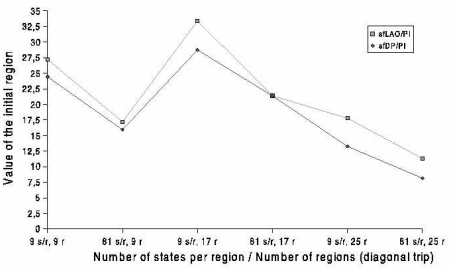
9 Résolution plus rapide, valeur plus faible

En suivant les idées précédentes, nous avons développé une autre version du procédé de *programmation dynamique focalisée* en faiblissant la condition d'arrêt de la fonction $Atteignable$, mais en suivant encore l'algorithme général 1. La nouvelle condition d'arrêt est obtenue dès qu'**au moins un état est atteint où les conditions de buts requises sont vérifiées** : $\mathcal{B} \cap E \neq \emptyset$ remplace $\mathcal{B} \subset E$. Nous proposons donc deux nouveaux algorithmes, *sfDP* et *sfLAO*, qui sont respectivement des versions «faibles» de *SFDP*, que nous avons développé, et *sLAO**, déjà existant. Les algorithmes *sfDP* et *sfLAO* ne sont manifestement pas optimaux, mais ils montrent des comportements incrémentaux intéressants, comme le montre la figure 7. Il est important de remarquer que les problèmes

résolus avec *sfDP* et *sfLAO* (figure 7) sont de tailles bien plus grandes que ceux résolus avec *SFDP* et *sLAO* (figure 6). Il apparaît que *sfLAO* trouve de meilleures solutions que *sfDP*, avec des temps de calcul similaires. Il est intéressant de remarquer que *sfLAO* montre le même comportement que *SFDP* : la qualité de la solution augmente avec le nombre de conditions de buts imposées à la solution optimale. Ainsi, le temps de calcul nécessaire à l'obtention de la solution optimale est également une fonction croissante qui atteint le temps de calcul obtenu avec *sLAO**, comme le montre la figure 7.

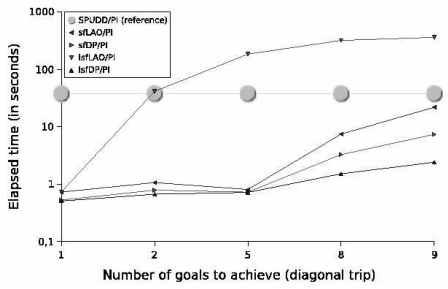


(a) Temps d'optimisation

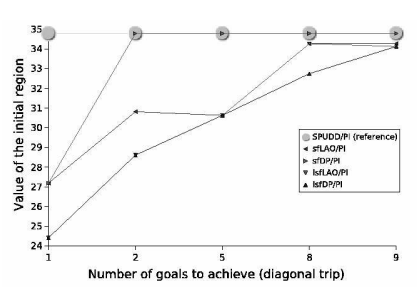


(b) Qualité de la solution

FIG. 7 – Comparaison entre *sfDP* et *sfLAO* en augmentant la taille du problème



(a) Temps d'optimisation



(b) Qualité de la solution

FIG. 8 – Comparaison sur la base de *SPUDD* (optimal) entre *IsfDP* et *IsfLAO* en augmentant le nombre de buts de planification à réaliser (9 états/région, 9 régions)

De ce fait, nous avons utilisé cette idée pour développer des versions incrémentales des algorithmes *sfDP* et *sfLAO* : les algorithmes *IsfDP* et *IsfLAO* décrits dans l'algorithme 2 et la figure 5.b. Les expérimentations (voir figure 8) montrent que *IsfDP* est clairement meilleur que *IsfLAO*. Des comparaisons précédentes (Teichteil-Königsbuch & Fabiani, 2005) entre des algorithmes de type *LAO** et de type *RTDP* ont montré que *LAO** serait meilleur quand la topologie du problème est «ouverte» et que *RTDP* serait plus efficace dans les «corridors». Ces deux procédés heuristiques mènent à limiter la taille de l'espace d'états exploré avant convergence. Nos algorithmes de type *SFDP* combinent ces deux approches, comme le montre la figure 5.b et ils améliorent rapidement la qualité de la solution. Cependant, ils ne mènent généralement pas à une

solution optimale à cause de la condition d'arrêt plus précoce dans l'étude d'atteignabilité. La preuve d'optimalité de *LAO** (Hansen & Zilberstein, 2001) s'appliquerait encore à *SFDP* si et seulement si : les états terminaux de chaque trajectoire dans l'espace d'états résultant de la *phase d'expansion* finissent dans l'ensemble des buts. Ceci est vérifié **si et seulement si** nous appliquons la condition d'arrêt de *sfLAO* dans les dernières *phases d'expansion* de l'algorithme, ce qui montre qu'un algorithme véritablement optimal peut être dérivé de *IsfDP* avec la condition suivante : la *phase d'expansion* s'arrête quand l'espace d'états se stabilise, indépendamment des buts atteints ou non.

Fonction IsfDP(L_b : **liste<DDB>** $\mathcal{E}, \mathcal{A}, \mathcal{I}$: **DDB** $\mathcal{T}, \mathcal{R}$: **DDA**) : $\mathcal{E}_k, \Pi_k, V_k$
  $\mathcal{E}_0 \longleftarrow \mathcal{I}$
  $k \longleftarrow 1$
  Tant que (L_b non vide) **faire**
    $\mathcal{I}_k \longleftarrow \mathcal{E}_{k-1}$
    $\mathcal{B}_k \longleftarrow$ tête de L_b
    Selon que
      $Algorithme = IsfDP : (\mathcal{E}_k, \Pi_k, V_k) \longleftarrow sfDP(\mathcal{E}, \mathcal{A}, \mathcal{I}_k, \mathcal{B}_k, \mathcal{T}, \mathcal{R})$
      $Algorithme = IsfLAO : (\mathcal{E}_k, \Pi_k, V_k) \longleftarrow sfLAO(\mathcal{E}, \mathcal{A}, \mathcal{I}_k, \mathcal{B}_k, \mathcal{T}, \mathcal{R})$
    Fin Selon que
    enlever la tête de L_b
    $k \longleftarrow k + 1$
  Fait
  Retourner $\mathcal{E}_k, \Pi_k, V_k$
Fin

Algorithme 2: Algorithmes *IsfDP* ou *IsfLAO* en utilisant l' algorithme générique 1 appliqué aux fonctions *sfDP* ou *sfLAO*

10 Conclusion

Nous avons proposé un algorithme original *SFDP* dont le temps d'optimisation et la qualité de la solution sont contrôlés à travers la définition de contraintes sur les buts de planification. Cependant, (Bonet & Geffner, 2003b) proposent un procédé général *TROUVE*-et-*REVISE* de programmation dynamique heuristique qui pourrait être confondu à tort avec notre procédé en deux phases : *expansion de l'espace d'états*-et-*programmation dynamique*. La conception de *SFDP*, grâce au procédé sous-jacent de *programmation dynamique focalisée*, rend possible la planification sous incertitudes dans des espaces d'états de grande taille pour des systèmes autonomes qui nécessitent l'obtention relativement rapide d'une solution, si possible optimale. Les conditions de buts peuvent être adaptées hors ligne ou en ligne, ce qui ouvre des directions intéressantes pour des travaux futurs sur la décision sous contraintes de temps et de ressources, et particulièrement pour notre application d'aéronef autonome. Nous avons par ailleurs développé un algorithme incrémental *IsfDP*, basé sur *sfDP*, qui est une version encore plus rapide mais plus faible de *SFDP*, permettant de réutiliser la solution sous-optimale obtenue avec *sfDP* dans un processus d'optimisation plus élaboré. *sfDP* trouve rapidement des solutions à des problèmes de grande taille, alors que la version incrémentale *IsfDP* améliore la solution courante grâce à des appels itératifs à *sfDP*, en augmentant la liste de sous-buts de planification à prendre en compte, ce qui est particulièrement

intéressant pour la replanification en ligne. Nous avons comparé, sur un ensemble de problèmes de tailles différentes, nos algorithmes de Programmation Dynamique avec Recherche Heuristique avec ceux déjà proposés dans la littérature. Nous allons à présent développer une version optimale du procédé de *programmation dynamique focalisée*, qui fournirait rapidement une solution courante pour des problèmes de grande taille, et qui serait améliorée vers la solution optimale globale si le temps est disponible.

Références

BACCHUS F., BOUTILIER C. & GROVE A. (1997). Structured solution methods for non-markovian decision processes. In *Proceedings 14th AAAI*, p. 112–117, Providence, RI.

BONET B. & GEFFNER H. (2003a). Faster heuristic search algorithms for planning with uncertainty and full feedback. In *IJCAI*, p. 1233–1238, Acapulco, Mexico.

BONET B. & GEFFNER H. (2003b). Labeled rtdp : Improving the convergence of real-time dynamic programming. In *Proceedings 13th ICAPS 2003*, p. 12–21, Trento, Italy.

BOUTILIER C., DEAN T. & HANKS S. (1999). Decision-theoretic planning : Structural assumptions and computational leverage. *J. of Artificial Intelligence Research*, **11**, 1–94.

BOUTILIER C., DEARDEN R. & GOLDSZMIDT M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, **121**(1-2), 49–107.

DEAN T. & KANAZAWA K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, **5(3)**, 142–150.

DEAN T. & LIN S.-H. (1995). Decomposition techniques for planning in stochastic domains. In *Proceedings 14th IJCAI*, p. 1121–1129, San Francisco, CA.

DEARDEN R. & BOUTILIER C. (1997). Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, **89**, 219–283.

FENG Z. & HANSEN E. (2002). Symbolic heuristic search for factored markov decision processes. In *Proceedings 18th AAAI*, p. 455–460, Edmonton, Alberta, Canada.

FENG Z., HANSEN E. A. & ZILBERSTEIN S. (2003). Symbolic generalization for on-line planning. In *Proceedings 19th UAI*, p. 209–216, Acapulco, Mexico.

HANSEN E. A. & ZILBERSTEIN S. (2001). Lao* : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, **129**, 35–62.

HAUSKRECHT M., MEULEAU N., KAELBLING L. P., DEAN T. L. & BOUTILIER C. (1998). Hierarchical solution of markov decision processes using macro-actions. In *Proceedings 14th UAI*, p. 220–229, San Francisco, CA.

HOEY J., ST-AUBIN R., HU A. & BOUTILIER C. (2000). *Optimal and Approximate Stochastic Planning using Decision Diagrams*. Rapport interne TR-2000-05, University of British Columbia.

PARR R. (1998). Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings 14th UAI*, p. 422–430, San Francisco, CA.

PUTERMAN M. L. (1994). *Markov Decision Processes*. John Wiley & Sons, INC.

TEICHTEIL-KÖNIGSBUCH F. & FABIANI P. (2004). Un modèle hybride en planification probabiliste d'exploration autonome. In *Proceedings 20th RFIA*, p. 1121–1129, Toulouse, CA.

TEICHTEIL-KÖNIGSBUCH F. & FABIANI P. (2005). Influence of modeling structure in probabilistic sequential decision problems. *RAIRO Operations Research*, **to appear**.

Classification automatique de données biographiques

Alexander Estacio-Moreno[1,2], Thierry Artières[1], Patrick Gallinari[1]

[1] LIP6, Université de Paris 6
{artieres, gallinari}@poleia.lip6.fr
[2] UR 079 - 013, Institut de Recherche pour le Développement,
32, ave. Henri Varagnat 93143 Bondy CEDEX
Alexander.Estacio-Moreno@bondy.ird.fr

Résumé : Nous présentons ici une méthode de classification de données biographiques basée sur l'estimation d'un mélange de densités à l'aide de modèles Markoviens. Afin de prendre en compte la durée de séjour dans les états nous proposons d'utiliser des modèles semi-markoviens dans lesquels différentes hypothèses de densités de durée peuvent être testées. Cette méthode, comme toute méthode de classification, manque toutefois de flexibilité. En effet, elle fournit un résultat unique de classification alors que l'on peut chercher à détecter, dans un même ensemble de données, des classifications de différentes natures. Nous proposons une méthode permettant de « guider » la classification en spécifiant partiellement le type d'information que l'on cherche à obtenir. Nous présentons pour cela une version modifiée de notre méthode de classification permettant d'introduire une connaissance a priori sur la similarité entre états. Nous fournissons des résultats expérimentaux d'une application à l'étude de la mobilité résidentielle.

Mots-clés : Apprentissage, Données biographiques, Mélange de densités, Modèles semi-markoviens, Fouille de données, Classification automatique.

1 Introduction

Les données biographiques sont issues d'enquêtes biographiques rétrospectives qui permettent saisir les mobilités. A partir de ces données on peut étudier finement des phénomènes qui reposent sur les comportements démographiques, économiques et sociaux, qu'ils soient individuels ou collectifs, et sur leur dynamique à différentes échelles spatiales et temporelles. Ces données décrivent des trajectoires définies par les changements d'état des variables socio-résidentielles, professionnelles, d'événements familiaux, etc., (GRAB (ed.), 1999).

Par l'analyse des données biographiques on cherche à décrire et relier entre elles les différentes formes de mobilité (résidentielle, socioprofessionnelle, …) pour comprendre leurs interactions et leurs impacts sur la réalité sociale. Les trajectoires décrites par ces données sont des séquences dont non seulement la dynamique et la

séquence des états sont des informations essentielles mais aussi les durées de séjour dans ces états.

Ces dernières années ont vu des avancées significatives dans l'analyse des données biographiques. En statistique on peut utiliser une approche modélisatrice, avec des modèles log-linéaires, des modèles logit et probit, des modèles de survie de Cox ... (Cox & Oakes, 1984 ; Courgeau & Lelièvre, 1989). Si l'on s'intéresse, par exemple, aux trajectoires socio-résidentielles d'une certaine population, cette approche permet de répondre à des questions du type : quels sont les déterminants de l'ascension socio-résidentielle ?

On peut également utiliser une approche exploratoire avec l'analyse typologique (ACP, AFC, cf. (Lebart *et al.*, 2002 ; Deville & Saporta, 1980)) qui permet, dans l'exemple antérieur, de répondre à des questions du type : Existe-il une (des) structure(s) dominante(s) dans les parcours socio-résidentiels de la population ?

Nous nous plaçons dans le second type d'approche. Nous abordons la classification de trajectoires comme étant un problème d'estimation de densité de probabilité, et nous proposons d'utiliser un mélange de densités. Cette approche permet d'obtenir une classification tout en fournissant de modèles pour chaque classe. Tout d'abord la dynamique et la séquence d'états des trajectoires sont modélisées par des chaînes de Markov. Ensuite, afin de prendre également en compte la durée de séjour dans les états nous utilisons des modèles semi-markoviens dans lesquels différentes hypothèses de densités de durée peuvent être testées. Puis, nous proposons une méthode permettant de « guider » la classification en spécifiant partiellement le type d'information que l'on cherche à obtenir. Nous présentons pour cela une version modifiée de notre méthode permettant d'introduire une connaissance a priori sur la similarité entre états. Enfin, nous fournissons des résultats expérimentaux d'une application à l'étude de la mobilité résidentielle, et une discussion en guise de conclusion.

2 Mélange de densités markoviennes

Un mélange de densités est une distribution de la forme (Bishop, 1995 ; Celeux, 1992) :

$$p(x) = \sum_{k=1}^{K} p(x/k)P(k) \qquad (1)$$

$$\text{Avec : } 1 < k < K \text{ , } 0 < P(k) < 1 \text{ et } \sum_{k=1}^{K} P(k) = 1$$

K étant le nombre de composantes du mélange, $P(k)$ les proportions du mélange (les probabilités a priori pour qu'une donnée x ait été générée par la composante k

du mélange), et $p(x/k)$ les densités composantes. Dans notre cas ces densités sont définies sur des séquences.

Soit $x_i = \{e_{i,1}, e_{i,2}, ..., e_{i,T_i}\}$ la séquence représentant la trajectoire de l'individu *i* ;

où $e_{i,t}$ est l'état à l'instant *t* de la trajectoire de l'individu *i*, et T_i est la longueur de la trajectoire de l'individu *i*, et l'on considère un espace d'états discrets avec $1 \le e_{i,t} \le m$.

Dans le cas des trajectoires socio-résidentielles, par exemple, l'instant *t* correspond à une année de résidence dans un endroit. La distribution de probabilité des individus est obtenue en récrivant (1) ainsi :

$$p(x_i/\Theta) = \sum_{k=1}^{K} p(x_i/\Theta_k) P(k) \qquad (2)$$

où Θ dénote l'ensemble des paramètres du modèle $\{P(1),...,P(K);\Theta_1,...,\Theta_K\}$. Etant donné *N* individus $X = (x_1, x_2, ..., x_N)$, le logarithme de la fonction de vraisemblance du mélange est :

$$L_M(\Theta) = \log p(X/\Theta) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} p(x_i/\Theta_k) P(k) \right) \qquad (3)$$

Le problème statistique consiste donc à apprendre trois ensembles de paramètres : les valeurs des $P(K)$ proportions et des Θ_K paramètres du mélange regroupés dans Θ, ainsi que le nombre de composantes *K*.

Considérons que les modèles pour chaque classe $p(x_i/\Theta_k)$ sont des modèles de Markov de premier ordre. Ces modèles servent à modéliser des séquences de changement d'état. Avec ce choix nous cherchons à nous approcher d'une façon simple à la dynamique des trajectoires biographiques.

Les paramètres du modèle pour chaque classe Θ_k sont : un vecteur de probabilité d'état initial $\pi_k(e)$ et une matrice *m x m* de probabilités de transition $a_k(e_t / e_{t-1})$. De la définition d'une chaîne de Markov on déduit :

$$p(x_i/\Theta_k) = \pi_k(e_{i,1}) \prod_{t=2}^{T_i} a_k(e_{i,t} / e_{i,t-1}) \qquad (4)$$

Pour effectuer la classification, on apprend les paramètres du mélange de densités par l'algorithme EM (Espérance-Maximisation, cf. Dempster et al., 1977). Le principe de cet algorithme est le suivant : on définit des paramètres de départ et ensuite à chaque itération, les valeurs de Θ sont ré-estimées de façon à accroître (3) et ceci jusqu'à ce qu'un maximum soit atteint. Voici l'algorithme pour le mélange de densités markoviennes :

Etape **E** : on calcule la probabilité a posteriori $p(i \in k / x_i, \Theta)$, c'est à dire la probabilité pour que la trajectoire x_i appartienne à la classe *k* sachant Θ, ou encore la probabilité que xi ait été générée par le modèle de la classe k.

$$p(i \in k / x_i, \Theta) = \frac{p(x_i / \Theta_k) P(k)}{\sum_{u=1}^{K} p(x_i / \Theta_u) P(u)} \; ; \; \forall i = 1,..,N ; \quad \forall k = 1,..,K \tag{5}$$

Etape **M** : on trouve des nouvelles valeurs des paramètres $\Theta^{Nouveau}$ qui maximisent le logarithme de la vraisemblance.

$$P(k)^{Nouveau} = \frac{1}{N} \sum_{i=1}^{N} p(i \in k / x_i, \Theta) \tag{6}$$

$$\pi_k^{Nouveau}(s) = \frac{\sum_{i=1}^{N} p(i \in k / x_i, \Theta) \delta(s, e_{i,1})}{\sum_{i=1}^{N} p(i \in k / x_i, \Theta)} \; ; \; \delta(s, e_{i,1}) = \begin{cases} 1 & Si \quad s = e_{i,1} \\ 0 & Sinon \end{cases} \tag{7}$$

$$a_k^{Nouveau}(s_q / s_p) = \frac{\sum_{i=1}^{N} p(i \in k / x_i, \Theta) r_i^{s_p \to s_q}}{\sum_{i=1}^{N} p(i \in k / x_i, \Theta) r_i^{s_p \to}} \tag{8}$$

$\forall k = 1,..,K$

où $r_i^{s_p \to s_q}$ est le compte des transitions depuis l'état s_p à l'état s_q dans la trajectoire de l'individu *i*, $r_i^{s_p \to}$ est le compte des transitions depuis l'état s_p à n'importe quel état dans la trajectoire de l'individu *i*, et $1 \le s, s_p, s_q \le m$.

Dans (Estacio-Moreno *et al.*, 2004) nous avons montré comment cette méthode fournit une classification de trajectoires migratoires.

3 Mélange de densités semi-markoviennes

Dans les données biographiques la durée passée dans les états est très importante. En effet, pour les trajectoires socio-résidentielles observées, par exemple, les durées de séjour varient selon les lieux de résidence (états du modèle). Cependant, les modèles de Markov classiques ne permettent pas de bien modéliser la durée passée dans un état donné. Dans un modèle de Markov la densité de durée : la probabilité de

rester une durée *d* dans l'état *e*, notée $p(d/e)$, suit une distribution exponentielle qui ne dépend que de a_{ee} (la probabilité de boucler dans l'état *e*) :

$$p(d/e) = (a_{ee})^{d-1}(1-a_{ee}) \tag{9}$$

Cette indépendance de la durée passée dans l'état *e* est une conséquence de la propriété markovienne (sans mémoire) : à chaque pas de temps le modèle détermine l'état suivant en ne se basant que sur l'état actuel.

Pour que le modèle rende compte de certains traits signifiants des trajectoires, par exemple pour les trajectoires socio-résidentielles : les durées de séjour dans certains espaces géographiques (une région, un quartier …), il est préférable d'expliciter d'une façon analytique la densité de durée $p(d/e)$ dans le modèle. (Ferguson, 1980) a été le premier en spécifier pour chaque état du modèle une densité de durée non paramétrique. Voir (Ramesh & Wilpon, 1992) pour un autre modèle non paramétrique. (Levinson, 1986 ; Russel & Moore, 1985) ont proposé des densités de durée paramétriques pour des modèles de Markov cachés. Nous allons spécifier une densité de durée paramétrique pour un modèle de Markov. Nous montrerons aussi comment calculer les paramètres de la densité de durée dans le cadre du mélange de densités.

Pour faire intervenir explicitement les durées associées aux états, on réécrit la trajectoire de l'individu *i*, x_i, ainsi :

$$x_i = \left\{\left(e_{i,1}, d_{i,1}\right), \left(e_{i,2}, d_{i,2}\right), \ldots \left(e_{i,NE_i}, d_{i,NE_i}\right)\right\} \tag{10}$$

où : $e_{i,j}$ est le $j^{ème}$ état de la trajectoire avec $e_{i,j} \neq e_{i,j-1}$, $d_{i,j}$ est la durée passée par l'individu *i* dans le $j^{ème}$ état de sa trajectoire (c'est-à-dire le nombre d'années pendant lesquelles l'individu *i* n'a pas changé de résidence, dans le cas de trajectoires socio-résidentielles) et NE_i est le nombre d'états de la trajectoire.

Donc, la vraisemblance d'une trajectoire conditionnée par son appartenance à une classe particulière Θ_k, est donnée par :

$$p\left(x_i/\Theta_k\right) = \pi_k\left(e_{i,1}\right) p_k\left(d_{i,1}/e_{i,1}\right) \prod_{j=2}^{NE_i} a_k\left(e_{i,j}/e_{i,j-1}\right) p_k\left(d_{i,j}/e_{i,j}\right) \tag{11}$$

On peut tester différentes lois de probabilité pour la durée de séjour. Supposons par exemple qu'elle suit une loi Log-normale, alors :

$$f\left(\mu, \sigma^2, d\right) = \frac{1}{d\left(2\pi\sigma^2\right)^{1/2}} \exp\left\{-\frac{\left(\log(d)-\mu\right)^2}{2\sigma^2}\right\} \tag{12}$$

$f\left(\mu, \sigma^2, d\right)$ est une loi continue à deux paramètres (μ et σ^2), mais les durées observées (dans les enquêtes rétrospectives) sont discrètes. Alors, pour obtenir la

densité de durée $p(d/e)$, $f(\mu,\sigma^2,d)$ peut être discrétisée sur l'intervalle des durées observées, $d_{\min} \le d \le d_{\max}$,ainsi :

$$p(d/e) = \frac{f(\mu(e),\sigma^2(e),d)}{\sum_{d'=d_{\min}}^{d_{\max}} f(\mu(e),\sigma^2(e),d')} \tag{13}$$

Maintenant, pour spécifier complètement le mélange de densités semi-markoviennes, pour chaque classe il faudra apprendre dans l'étape M de l'algorithme EM, en plus des paramètres des équations (6, 7, 8), les valeurs des paramètres μ et σ^2 . Nous donnons ci-dessous les équations pour effectuer ces calculs.

$$\mu_k^{Nouveau}(s) = \frac{\sum_{i=1}^{N} p(i \in k / x_i,\Theta) \sum_{j=1}^{NS_i^s} \log(d_{i,j}^s)}{\sum_{i=1}^{N} p(i \in k / x_i,\Theta) NS_i^s} \tag{14}$$

$$\sigma_k^{2\,Nouveau}(s) = \frac{\sum_{i=1}^{N} p(i \in k / x_i,\Theta) \left[\sum_{j=1}^{NS_i^s} (\log d_{i,j}^s - \mu_k(s))^2 \right]}{\sum_{i=1}^{N} p(i \in k / x_i,\Theta) NS_i^s} \tag{15}$$

$\forall k = 1,..,K \;\; ; \forall s = 1,..,m$.

où $d_{i,j}^s$ est la durée du $j^{ème}$ séjour de l'individu *i* dans l'état *s* et NS_i^s est le nombre de séjours de l'individu *i* dans l'état *s*.

Ce cadre est assez général pour permettre de tester différentes hypothèses de lois modélisant la probabilité de séjour dans les états, dans un mélange de densités semi-markoviennes.

L'apprentissage du nombre de classes est un problème ouvert. Des nombreux critères proposent de pénaliser le maximum de log-vraisemblance par une fonction (souvent linéaire) du nombre de dégrées de liberté pour contrer sa croissance systématique avec ce paramètre. BIC (*Bayesian Information Criterion*. Schwarz, 1978) et AIC (*An Information Criterion*. Akaike, 1973), appartiennent à cette catégorie de critères. Le mélange de densités ne satisfait pas les conditions de régularité de BIC (Fraley & Raftery, 2000), et aucun consensus n'existe sur la valeur du coefficient pénalisant la vraisemblance dans le critère AIC (Biernacki, 1997). Cependant, ces critères se sont montrés appropriés, ayant des bonnes performances, pour une variété d'applications dans le cadre du mélange de gaussiennes.

D'autres critères privilégient le choix d'un seuil de diminution de la croissance dans la courbe du maximum de la vraisemblance pour trouver le nombre de classes. (Cutler & Windham, 1993) proposent une procédure pour déterminer le coude de la vraisemblance : le critère EL (*Elbow Likelihood*). Voir (McLachlan & Peel, 2000) pour une révision de critères pour la détermination du nombre de classes.

4 Prise en compte d'information a priori

La classification avec un mélange de densités markoviennes (ou semi-markoviennes), comme toute méthode de classification automatique, manque de flexibilité. En effet, cette méthode ne peut fournir qu'un résultat unique de classification alors que l'on peut chercher à identifier, pour un même ensemble de données, des classifications exprimant différents types de tendances ou catégories.

Dans le but d'obtenir des meilleurs résultats nous allons modifier notre cadre méthodologique pour permettre de « guider » la classification. Nous chercherons à enrichir l'algorithme à partir d'une connaissance a priori qu'on pourrait avoir sur les données.

On peut spécifier de façon naturelle la connaissance a priori par un critère de similarité entre les états de la variable analysée. Si l'on considère qu'un état e_1 est très similaire à un état e_2 l'algorithme d'apprentissage peut être modifié de façon à ce que cette similarité soit prise en compte. Si on observe par exemple une transition de l'état e_1 à un autre état e_3 on peut alors considérer qu'on a aussi observé, d'une certaine façon, une transition de l'état e_2 à l'état e_3.

Considérons le cas de notre variable socio-résidentielle exprimant les changements de lieux de résidence (états du modèle). Un critère de similarité entre états pourrait être défini à partir du niveau socioéconomique des états ou bien de leurs distances au centre-ville. Le choix du critère dépend de la nature de la typologie que l'on souhaite découvrir. En guidant la classification on s'intéresse à l'obtention d'une part, des classifications différentes selon le critère de similarité choisi (socioéconomique ou distance) et d'autre part, des classes plus homogènes au sens d'un même critère.

Nous avons considéré une similarité définie entre transitions plutôt qu'entre états car cela corresponde plus à ce que l'on souhaite découvrir en général dans les trajectoires biographiques (qu'elles soient socio-résidentielles, professionnelles, …). Dans la suite, $Sim(s_p, s_q, \nu, \mu)$ désigne la similarité entre une transition de l'état s_p à l'état s_q et une transition de l'état ν à l'état μ. Ces similarités sont fixées à la main à partir des connaissances a priori sur le domaine et du type de catégorisation que l'on souhaite obtenir. Elles sont prises en compte dans l'apprentissage pour la réestimation des probabilité de transition $a_k(s_q / s_p)$.

On rajoute dans l'étape M de l'algorithme EM, une fois que les probabilités de transitions ont été réestimées de façon classique, une seconde réestimation de ces probabilités tenant compte des similarités[1] :

$$\hat{a}_k(s_q / s_p) = \frac{\sum_{\nu,\mu} Sim(s_p, s_q, \nu, \mu) \quad a_k(\mu / \nu)}{\sum_{\nu,\mu} Sim(s_p, s_q, \nu, \mu)} \tag{16}$$

En introduisant une étape additionnelle de réestimation (R) dans l'algorithme EM, on cherche dans chaque classe, d'un coté, à renforcer les transitions entre états plus similaires, et d'un autre coté, à affaiblir les transitions entre états moins similaires.

Cette réestimation a pour effet de lisser l'estimation des probabilités de transitions en forçant des transitions similaires à avoir des probabilités similaires. Au fil de l'apprentissage, cela focalise les modèles des classes sur différents types de trajectoires faisant intervenir les mêmes types de transitions. Cette étape agit à la manière d'un a priori sur la nature des lois de transitions apprises par les modèles.

5 Expériences : les choix d'analyse

Les données que nous utilisons proviennent d'une enquête biographique effectuée à Cali (Colombie) en 1998 (Barbary, 2001). Nous ne nous sommes intéressés ici qu'aux informations concernant la mobilité socio-résidentielle à Cali. Voir l'annexe 1 pour les modalités (états) de la variable socio-résidentielle utilisée.

Nous avons utilisé un temps d'analyse biographique, c'est-à-dire que nous avons choisi un évènement de référence (le moment de l'enquête : 1998), et nous avons considéré les 59 années (période d'analyse) antérieures à cet évènement. Pour prendre en compte la censure (les individus entrant dans l'analyse après le début de la période d'analyse) la modalité zéro a été ajoutée à la variable socio-résidentielle (Voir l'annexe 1).

L'approché a été appliquée sur un échantillon de 1768 individus. Quatre modèles différents ont été utilisés en tant que densités du mélange : un modèle de Markov (M) et trois modèles semi-markoviens avec des probabilités de durée Normale (SMN), Poisson (SMP) et Log-Normale (SMLN). Une information a priori, établit par des experts du domaine, a été introduite dans l'algorithme. Cette information cherchait à rendre compte de la similarité entre états en termes de leur niveau socioéconomique.

[1] En pratique, l'intensité de la similarité entre transitions est contrôlée par un paramètre α de la façon suivante : $Sim^{\alpha}(s_p, s_q, \nu, \mu)$. Plus α est grand, moins la similarité influence le calcul de $\hat{a}_k$.

Pour chaque modèle nous avons effectué dix initialisations aléatoires sur les probabilités a posteriori. L'algorithme s'arrête dès qu'un maximum du logarithme de la vraisemblance est atteint. Pour chaque modèle, parmi toutes les différentes classifications résultantes nous conservons celle dont la valeur du logarithme de la vraisemblance finale est la plus élevée.

6 Résultats

L'évaluation de méthodes de classification est un problème ouvert. Sachant que l'un des buts des travaux sur ce type de données est de fournir des résultats humainement faciles à interpréter, nous avons cherché à utiliser quelques mesures objectives. Nous avons défini une mesure reflétant la facilité d'interprétation des classes : l'Homogénéité Intra-classe, notée HI. Cette mesure permet d'évaluer la cohésion des individus dans les classes.

$$HI_k = \frac{\sum_{i \in k} \delta_i}{n_k} \text{ ; où } \begin{cases} \delta_i = 0 \quad si \quad p(i \in k / x_i, \Theta) < 0{,}5 \\ \delta_i = 1 \quad si \quad p(i \in k / x_i, \Theta) \geq 0{,}5 \end{cases} \tag{17}$$

Egalement, nous mesurons l'instant de sortie de la censure (noté ISC) d'au moins 50 % des individus de chaque classe. C'est le moment à partir duquel au moins 50% d'individus d'une classe sont dans des états socio-résidentiels autres que l'état de censure. Cette mesure par classe permet d'analyser l'ensemble des classes selon la longueur des trajectoires.

$$ISC_k = t / P_k(censurés, t) < 0{,}5 \tag{18}$$

où $P_k(censurés, t) = \frac{nb\ d'individus\ censurés\ en\ t}{n_k}$, $1 \leq t \leq 59$ et n_k est le nombre d'individus de la classe *k*.

Par ailleurs, de façon plus subjective, nous avons caractérisé avec des experts du domaine les différents états selon leur niveau socioéconomique. Cette caractérisation nous a permis de déterminer si les trajectoires sont descendantes, stables ou ascendantes, à partir de la succession de leurs états. Du point de vue des experts une classification devrait aboutir à des classes homogènes de ce point de vue là. Nous nous servirons de cette étiquetage des données pour évaluer si l'information a priori introduite dans l'algorithme permet bien de respecter ce type de contraintes.

Les critères AIC et BIC ont été envisagés pour la détermination automatique du nombre de classes, mais ils se sont montrés inappropriés pour les densités composantes que nous utilisons (markoviennes et Semi-markoviennes Normal, Poisson et Log-normal) : ils tendent à choisir un nombre de classes trop petit. Dans nos expériences, ces deux critères classent les 1768 trajectoires au maximum dans

trois classes, ce qui est très étonnant vu la taille de l'espace d'états et la diversité des trajectoires de l'échantillon. Le nombre de classes (10) a donc été déterminé par le critère du coude de la vraisemblance (EL).

Les tables 1 e 2 montrent les résultats pour les mesures HI et ISC. Les dernières lignes de ces tables correspondent à la HI globale et à l'écart type de l'ISC. Globalement, une meilleure cohésion des individus aux classes est obtenue lorsqu'on introduit explicitement la durée de séjour. La définition d'une loi pour les durées de séjour des individus dans les états, fait qu'ils sont mieux attachés à leurs classes. De plus, on observe une meilleure différentiation des classes selon la longueur moyenne des trajectoires : les ISC sont très proches dans la classification obtenue avec des modèles sans durée explicite (M a un écart type très petit), et plus distants dans celles obtenues par des modèles avec durée explicite (les écart types sont beaucoup plus élevés).

A partir de la table 3 on peut voir que globalement les trajectoires sont stables ou ascendantes. Aussi bien sans similarité (SS) qu'avec similarité (AS) les pourcentages les plus élevés sont observés pour ces deux types de trajectoires. Par ailleurs, la prise en compte de la similarité permet, d'une part, de faire ressortir une classe additionnelle de trajectoires stables (classe 7) et, d'autre part, de mieux différentier les classes par la composition majoritaire des types de trajectoires qu'elles regroupent : avec l'intégration d'information a priori 6 classes contiennent plus de 60% de trajectoires de même tendance contre 4 classes sans l'utilisation d'information a priori.

Table 1. Homogénéité Intra-classe

HI				
Classe	M	SMN	SMP	SMLN
1	0,86	0,95	0,98	0,96
2	1,00	0,95	1,00	0,91
3	0,75	0,99	0,94	0,94
4	0,85	1,00	1,00	0,98
5	0,65	0,99	1,00	0,91
6	0,82	0,97	0,99	0,94
7	0,98	1,00	0,94	0,92
8	0,71	1,00	0,99	1,00
9	0,95	0,99	0,96	1,00
10	0,84	0,99	1,00	0,93
HIG	0,78	0,98	0,98	0,95

Table 2. Instant de sortie de censure

ISC				
Classe	M	SMN	SMP	SMLN
1	36	18	46	36
2	47	47	34	47
3	40	35	45	49
4	39	51	10	25
5	36	21	31	28
6	45	45	19	42
7	36	47	36	51
8	49	55	53	51
9	43	32	31	55
10	37	41	20	40
ET	4,89	12,46	13,43	10,16

Table 3. Pourcentage de trajectoires descendantes, stables et ascendantes par classe, pour la classification avec une densité de durée suivant une loi Normale.

Classes		1	2	3	4	5	6	7	8	9	10
Desc.	SS	14,50	19,88	18,57	21,21	21,32	11,11	14,29	14,02	19,17	17,58
	AS	16,62	13,73	13,64	19,83	0,00	11,33	5,71	21,83	19,01	16,37
Stbl.	SS	48,61	24,22	15,71	16,67	15,44	11,67	40,14	28,97	28,33	28,48
	AS	46,29	40,24	22,73	12,40	0,00	14,00	74,29	23,14	30,99	16,96
Asce.	SS	36,89	55,90	65,71	62,12	63,24	77,22	45,58	57,01	52,50	53,94
	AS	37,08	46,02	63,64	67,77	100,0	74,67	20,00	55,02	50,00	66,67

7 Conclusion

Nous avons présenté une méthode de classification de trajectoires biographiques basée sur l'estimation d'un mélange de densités. La dynamique et la séquence d'états ont été modélisées par des chaînes de Markov. Nous avons étendu cette première approche pour prendre en compte la durée de séjour dans les états. Les densités composantes deviennent des modèles semi-markoviens. Nous avons montré que les paramètres du mélange de densités semi-markoviennes peuvent être appris avec l'algorithme EM. La prise en compte des durées de séjour permet d'obtenir des classes plus stables (des individus mieux attachés aux classes) et mieux séparées selon la longueur de trajectoires. Il est apparu que les modèles semi-markoviens permettent d'obtenir des meilleurs résultats que les modèles markoviens (sans durée explicite). Cette nouvelle approche est applicable à n'importe quel type de données séquentielles où la durée de séjour dans les états est importante. Nous avons également proposé d'introduire une information a priori dans d'apprentissage de façon à orienter la nature des classes découvertes. Nous avons montré que la prise en compte d'une information a priori de ce type permet d'améliorer les résultats en homogénéisant les classes au sens du critère de similarité.

Références

AKAIKE, H. (1973). *Information Theory as an Extension of de Maximun Likelihood Principle*, In V. Petrov and F. Csaki. Editors, Second International Symposium on Information Theory, Budapest, Akademiai Kiado, pp. 267-281.

BARBARY, O. (2001). *Mesure et Réalité de la Segmentation Socio-raciale : Une Enquête sur les Ménages Afrocolombiens à Cali*, Population, vol. 56 n°5, Paris, pp. 773-810.

BIERNACKI, C. (1997). *Choix de Modèles en Classification*, Thèse de doctorat, Université de Technologie de Compiègne,195p.

BISHOP C. M. (1995). *Neural Networks For Pattern Recognition*, Oxford University Press, New York, 482 p.

CELEUX G. (1992). *Modèles Probabilistes en Classification*. In Modèles pour l'analyse des Données Multidimensionnelles. Editeurs J.J DROESBEKE B. FICHET & PH.TASSI; éditions Economica, pp. 165-214.

COURGEAU D. & LELIEVRE E. (1989). *Analyse Démographique des Biographies*, INED, Paris, 268 p.

COX D.R. & OAKES D. (1984). *Analysis of Survival Data*, Chapman y Hall, Londres, 201 p.

DEMPSTER, A.P. & LAIRD, N.M. & RUBIN D. B. (1977). *Maximum Likelihood from Incomplete data via the EM algorithm*. Journal of the Royal Statistical Society, Series B, n° 34, pp.1-38.Analysis of survival data, Chapman y Hall, Londres, 201 p.

DEVILLE, J.C. & SAPORTA, G. (1980). *Analyse Harmonique Qualitative*, in Data Analysis and Informatics, E. DIDAY et al. éditeurs, North Holland Publishing Compagny, pp. 375-389.

ESTACIO-MORENO, A. & BARBARY, O. & GALLINARI P. & PIRON M. (2004). *Classification de Données Biographiques : Application à des Trajectoires Migratoires vers Cali (Colombie)*. In Revue de Statistique Appliquée, vol. LII (4). pp. 33-54.

FERGUSON J. D. (1980). *Variable Duration Models for Speech*. In J.D. FERGUSON, editor, Proc. Symposium on the Application of Hidden Markov Models to Text and Speech, Princeton, NJ. pp 143-179.

FRALEY C. & RAFTERY A. E. (2000). *Model-Based Clustering, Discriminant Analysis, and Density Estimation*. Working Paper N° 11. Center for Statistics and Social Sciences, University of Washington, 46p.

GRAB, (ed.) (1999). *Biographies d'Enquêtes, Bilan de 14 Collectes Biographiques*, Paris, INED-PUFF. Collection Méthodes et savoirs n° 3, 340 p.

LEBART, L. & MORINEAU, A. & PIRON, M. (2002). *Statistique Exploratoire Multidimensionnelle*. Paris, 2002, éd. Dunod, 437p.

LEVINSON, S.E. (1986), *Continuously Variable Duration Hidden Markov Models for Automatic Speech Recognition*, Computer Speech and Language, 1, pp. 29-45.

MACLACHLAN, G. & PEEL D. (2000), *Finite Mitures Models*, John Wiley and Sons, Inc.

RAMESH, P. & WILPON, J. (1992). *Modeling State Durations in Hidden Markov Models for Automatic Speech Recognition*. In Proc. ICASSP, pages 381-384.

RUSSEL, M.J. & MOORE, R.K. (1985). *Explicit Modeling of State Occupancy in Hidden Markov Models for Speech Signals*, in Proceedings ICASSP-85 International Conference on Acoustics, Speech, and Signal Processing, pp. 5 - 8.

SCHWARZ, G. (1978), *Estimating the Dimension of a Model*, Annals of Statistics, 6 : 461-464.

Annexe 1. Etats de la variable socio-résidentielle

La variable socio-résidentielle a été construite à partir de variables originales de l'enquête décrivant : la zone géographique de la ville où se trouve la résidence, sa strate socioéconomique, le statut d'occupation de la résidence et la corésidence de l'individu avec ses parents. Le tableau A1 montre les modalités de ces variables.

Table A1. Variables pour l'analyse de la mobilité socio-résidentielle

Zone géographique	Strate Socioéconomique	Statut d'Occupation de la résidence et corésidence de l'individu avec ses parents
1 Est	1. Populaire	1. Usufruit
2 Centre-Est	2. moyenne	2. Propriétaire ou conjoint du propriétaire
3 Couloir Central	3. Aisée	3. Enfant ou petit enfant du propriétaire ou de son conjoint
4 Ouest		4. Hébergé du propriétaire ou de son conjoint
		5. Locataire ou conjoint du locataire
		6. Enfant ou petit enfant du locataire ou de son conjoint
		7. Hébergé du locataire ou de son conjoint

Les modalités de la variable socio-residentielle représentant les états des chaînes markoviennes (ou semi-markviennes) son données ci-dessous :

0 Censure à gauche[2]
1 Hébergé dans des secteurs populaires.
2 Usufruitier dans des secteurs populaires de l'Est.
3 Usufruitier dans d'autres secteurs populaires.
4 Enfant ou petit enfant du locataire ou de son conjoint, dans des secteurs populaires de l'Est.
5 Enfant ou petits enfant du locataire ou de son conjoint, dans d'autres secteurs populaires.
6 Locataire ou conjoint du locataire, dans des secteurs populaires de l'Est.
7 Locataire ou conjoint du locataire, dans des secteurs populaires du Centre-Est.
8 Locataire ou conjoint du locataire, dans d'autres secteurs populaires.
9 Enfant ou petit enfant du propriétaire ou de son conjoint, dans des secteurs populaires de l'Est.
10 Enfant ou petit enfant du propriétaire ou de son conjoint, dans d'autres secteurs populaires.

[2] Cet état additionnel sert à prendre en compte la partie de la trajectoire des individus entrant dans l'analyse après le début de la période d'analyse.

11 Propriétaire ou conjoint du propriétaire, dans des secteurs populaires de l'Est.

12 Propriétaire ou conjoint du propriétaire, dans des secteurs populaires du Centre-Est.

13 Propriétaire ou conjoint du propriétaire, dans d'autres secteurs populaires.

14 Usufruitier ou hébergé dans des secteurs de classe moyenne.

15 Enfant ou petit enfant du locataire ou de son conjoint, dans des secteurs de classe moyenne.

16 Locataire ou conjoint du locataire, dans des secteurs de classe moyenne de l'Est ou du Centre-Est.

17 Locataire ou conjoint du locataire, dans des secteurs de classe moyenne de l'Ouest ou du Couloir Central.

18 Enfant ou petit enfant du propriétaire ou de son conjoint, dans des secteurs de classe moyenne.

19 Propriétaire ou conjoint du propriétaire, dans des secteurs de classe moyenne de l'Est ou du Centre-Est.

20 Propriétaire ou conjoint du propriétaire, dans des secteurs de classe moyenne de Ouest ou du Couloir Central.

21 Usufruitier ou hébergé dans des secteurs de classe aisée du Couloir Central.

22 Enfant ou petit enfant du locataire ou de son conjoint, locataire ou conjoint du locataire, dans des secteurs de classe aisée du Couloir Central ou d'Ouest.

23 Enfant ou petit enfant du propriétaire ou de son conjoint, propriétaire ou conjoint du propriétaire, dans des secteurs de classe aisée du Couloir Central ou d'Ouest.

24 Hors Cali.

Étiquetage sémantique flou de documents XML

Gaëlle Hignette

UMR INA P-G/INRA MIA, Institut National Agronomique de Paris-Grignon,
16 rue Claude Bernard, 75231 Paris Cedex 05
hignette@inapg.fr

Résumé :
Ce travail s'inscrit dans le cadre de la construction semi-automatique d'un entrepôt de données XML ouvert sur le web, appliqué au domaine du risque alimentaire. Une ontologie permet d'annoter les documents provenant du web et de guider leur interrogation. Nous nous intéressons plus particulièrement à l'annotation de tableaux issus de publications scientifiques. Un terme provenant du web peut être rapproché de plusieurs termes de l'ontologie mais chaque rapprochement est incertain. Les annotations sont représentées sous forme de distributions de possibilités, associant aux termes de l'ontologie la possibilité qu'ils représentent le terme du web. Ce papier présente deux calculs du degré de possibilité : le premier est fondé sur une comparaison de mots, les mots dans les termes de l'ontologie étant pondérés suivant leur importance sémantique ; le second calcul s'appuie sur le premier mais prend également en considération les relations de spécialisation définies dans l'ontologie.

Mots-clés : ontologies, extraction de connaissance, enrichissement sémantique, logique floue, XML.

1 Introduction

Le travail que nous présentons dans cet article est mené dans le cadre du projet e.dot, pour Entrepôt de Données ouvert sur la Toile (e.dot, 2004). Ce projet consiste à construire automatiquement un entrepôt thématique de données alimenté par des données extraites du web. Le domaine d'application choisi est la prévention du risque microbiologique dans les aliments. La construction de l'entrepôt de données repose sur une ontologie. Cette ontologie s'appuie sur trois taxonomies (ensembles de termes structurés selon une hiérarchie de spécialisation), représentant les produits alimentaires, les microorganismes et les facteurs étudiés susceptibles de modifier le comportement des microorganismes. Ces taxonomies sont augmentées dans l'ontologie par la définition de relations (au sens base de données) qui peuvent être construites sur les termes : par exemple, la relation *AlimentPH*, qui associe à un type d'aliment son pH. Selon la classification proposée par (Guarino, 1998), il s'agit donc d'une *ontologie d'application*, qui s'appuie sur deux *ontologies de domaine* : la microbiologie (microorganismes et

facteurs étudiés) et l'alimentation, en y ajoutant des informations spécifiques au risque alimentaire. Dans cet article, nous n'exploitons que la partie taxonomie de l'ontologie.

L'ontologie est centrale dans la mise en place de notre entrepôt de données thématique. En effet, elle fournit les termes d'amorce qui permettent d'effectuer une recherche pour récupérer des documents sur le web. La fréquence d'apparition des termes de l'ontologie dans les documents récupérés permet également un filtrage de ces documents afin de sélectionner les documents pertinents à conserver dans l'entrepôt. L'ontologie est finalement utilisée pour annoter les documents, qui seront interrogés sur les termes de cette même ontologie. Le format de représentation des données choisi est XML. L'avantage de ce format est sa flexibilité, qui permet de prendre en compte l'hétérogénéité des sources de données que l'on trouve sur le web.

L'ontologie utilisée pour construire l'entrepôt de données XML a été mise en place au cours du projet Sym'Previus (Sym'Previus, 2004). Cette ontologie permet l'interrogation par le système MIEL (Buche & Haemmerlé, 2000; Buche *et al.*, 2003) d'une base de données hétérogènes en microbiologie alimentaire : la complexité de la base est cachée à l'utilisateur, qui sélectionne simplement dans l'ontologie les microorganismes, produits alimentaires et facteurs qui l'intéressent et reçoit les résultats de sa requête sous forme tabulaire. Dans le cadre du projet e.dot, nous utilisons la même ontologie pour annoter les données provenant du web, afin de pouvoir interroger l'entrepôt de données au format XML en même temps que la base existante, de manière transparente pour l'utilisateur. Le système d'interrogation MIEL étendu à l'entrepôt de données XML, appelé MIEL++, devra présenter les mêmes avantages que le système MIEL actuel : il devra notamment permettre à l'utilisateur d'exprimer des préférences dans ses critères de sélection afin d'ordonner les réponses suivant leur adéquation à la requête.

Pour enrichir l'entrepôt de données, nous avons choisi dans un premier temps de nous concentrer sur les documents contenant des tableaux de données. En effet, de nombreuses publications en microbiologie alimentaire contiennent des tableaux qui présentent les résultats expérimentaux : cette information est synthétique et fiable, et la forme tabulaire des données en facilite le traitement.

Dans cet article, nous considérons que la forme des tableaux a été reconnue au préalable : ce travail est mené par d'autres membres du projet e.dot (e.dot, 2004). Nous nous intéressons à l'annotation sémantique de ces tableaux en utilisant les termes de l'ontologie, dans le but de pouvoir interroger l'entrepôt via ces termes. Nous proposons une annotation sémantique floue des termes du tableau par rapport aux termes de l'ontologie : chaque terme du tableau est représenté par des termes de l'ontologie ordonnés suivant leur pertinence par rapport au terme initial.

Dans cet article, nous présentons tout d'abord des rappels sur la théorie des sous-ensembles flous que nous utilisons dans notre système d'annotations, ainsi que le formalisme choisi pour l'annotation des tableaux. Nous étudions ensuite les deux scores qui permettent d'ordonner les termes de l'ontologie suivant leur pertinence par rapport aux termes du tableau : dans la partie 3 nous présentons le score lexical de ressemblance, dans lequel les termes de l'ontologie sont considérés isolément les uns des autres, puis nous présentons dans la partie 4 le score de ressemblance hiérarchique, qui permet de prendre en compte la hiérarchie qui structure l'ontologie. Dans chacune de ces deux parties, nous évaluons l'intérêt de ces deux scores par une approche expérimentale.

2 Prérequis

Le système d'annotations que nous souhaitons mettre en place doit pouvoir représenter différentes propositions pour chaque terme à annoter, avec un ordre de pertinence. Il faut que ces annotations puissent être comparées à une expression de préférences dans la requête d'un utilisateur. Nous présentons ici la théorie des sous-ensembles flous, qui s'adapte à cette problématique, puis la formalisation choisie pour les annotations.

2.1 Sous-ensembles flous

Nous utilisons ici la notion de sous-ensemble flou introduite par (Zadeh, 1965) et prolongée par la théorie des possibilités (Zadeh, 1978).

2.1.1 Définition d'un sous-ensemble flou

La notion de sous-ensemble flou est un assouplissement de la notion de sous-ensemble classique d'un ensemble de référence X. Dans le cas classique, les éléments de X qui possèdent une certaine propriété constituent un sous-ensemble A de X, les éléments de X qui ne possèdent pas cette propriété appartiennent au complémentaire de A dans X. Dans le cas d'un sous-ensemble flou, les éléments peuvent appartenir partiellement à un sous-ensemble, avec un degré d'appartenance compris entre 0 (élément n'appartenant pas au sous-ensemble) et 1 (élément appartenant totalement au sous-ensemble).

Définition 1
Un sous-ensemble flou A *d'un ensemble de référence* X *est défini par une fonction d'appartenance* μ_A *de* X *dans* $[0, 1]$ *qui associe à chaque élément* x *de* X *le degré* $\mu_A(x)$ *avec lequel* x *appartient à* A*.*

2.1.2 Appariement entre critère de sélection flou et donnée imprécise

Les sous-ensembles flous utilisés dans Sym'Previus ont deux rôles distincts : la représentation de l'imprécision dans les données et la représentation de préférences dans les requêtes (critère de sélection flou).

Définition 2
Le degré de possibilité d'adéquation *ou* degré d'intersection flou *entre deux sous-ensembles flous* A *et* B *de* X*, l'un représentant un critère de sélection flou et l'autre une donnée imprécise, ayant respectivement pour fonction d'appartenance* μ_A *et* μ_B*, est* $\pi(A, B) = sup_{x \in X}(min(\mu_A(x), \mu_B(x)))$*.*

Exemple 1
La figure 1.A présente un critère de sélection flou pour une personne qui veut obtenir des réponses sur le lait demi-écrémé, mais n'exclut pas que des résultats sur le lait entier ou écrémé puissent l'intéresser dans une moindre mesure. La figure 1.B présente une donnée imprecise : le terme « low fat milk » d'une publication anglophone signifie le plus probablement du lait écrémé, mais peut-être aussi du lait demi-écrémé. Le degré

de possibilité d'adéquation entre la requête et la donnée est de 0.5 (valeur obtenue sur lait demi-écrémé*).*

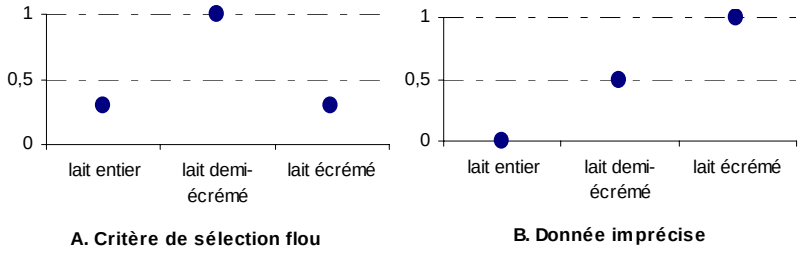


FIG. 1 – Deux sous-ensembles flous définis sur l'ensemble de valeurs discret {lait entier, lait demi-écrémé, lait écrémé}

2.2 Le format SML

Dans ce travail, les données qui nous intéressent sont représentées dans des publications scientifiques sous forme de tableaux.

Le format SML (Semantic Markup Language) a été proposé par (Saïs *et al.*, 2005) pour représenter en XML des tableaux, enrichis sémantiquement par des termes et relations d'une ontologie. Le tableau est représenté sous forme d'un ensemble de lignes. Chaque ligne du tableau est découpée en autant de relations que l'on peut en reconnaître : les relations sont reconnues grâce aux types des colonnes du tableau. En effet, chaque relation définie dans l'ontologie peut être identifiée par sa signature. Tout concept de l'ontologie participant à une signature de relation est un candidat pour le type d'une colonne : si une majorité des éléments de la colonne, ou son titre, correspond à ce concept, alors ce concept sert de type pour la colonne. L'ensemble des types des colonnes du tableau est comparé avec chaque signature de relation, afin de trouver quelles relations sont représentées dans le tableau (Saïs *et al.*, 2005).

Dans chaque ligne et pour chaque relation, chaque attribut est représenté par le terme d'origine (valeur d'une case du tableau) et par le ou les termes qu'on a pu lui associer dans l'ontologie de référence. Les termes de l'ontologie associés aux termes d'origine sont obtenus par inclusion ou intersection de mots. Ils constituent l'annotation sémantique du tableau ; ce sont ces valeurs qui seront utilisées comme valeurs de sélection lors de l'interrogation des données.

Exemple 2

Le tableau de données présenté dans la figure 2 comporte deux colonnes. Le type de la première colonne est reconnu comme Aliment, celui de la deuxième colonne comme pH. Ces deux types composent la signature de la relation AlimentPH. Chaque ligne du tableau (balise ligneRel*) contient donc une seule relation,* AlimentPH*. Chaque relation AlimentPH a deux attributs,* Aliment *et* pH*. Pour chaque Aliment, le terme que l'on trouve dans le tableau est présenté dans la balise* origineVal *; l'annotation sémantique consiste en l'ajout des termes de l'ontologie (balises* finalVal*) qui représentent le terme d'origine.*

Produit	pH
Fromage de chèvre	6.6
Oignon rouge	5.2

Figure 2.A - pH des produits utilisés

```
<table> <titre> pH des produits utilisés </titre>
<titre-col> produit </titre-col>
<titre-col> pH </titre-col> <contenu>
<ligneRel> <AlimentPH>
  <Aliment>
    <origineVal> Fromage de chèvre </origineVal>
    <finalVal> fromage </finalVal>
    <finalVal> lait de chèvre </finalVal>
    <finalVal> viande de chèvre </finalVal>
  </Aliment>
  <pH> <origineVal> 6.6 </origineVal> </pH>
</AlimentPH> </ligneRel>
<ligneRel> <AlimentPH>
  <Aliment>
    <origineVal> Oignon rouge </origineVal>
    <finalVal> oignon d'Egypte </finalVal>
    <finalVal> oignon de printemps </finalVal>
    <finalVal> chou rouge </finalVal>
  </Aliment>
  <pH> <origineVal> 5.2 </origineVal> </pH>
</AlimentPH> </ligneRel> </contenu> </table>
```

FIG. 2 – Un tableau de données et sa représentation simplifiée en SML

2.3 Représentation du flou en XML

Lors de la représentation de tableaux en SML, il peut y avoir plusieurs annotations sémantiques (balises *finalVal*) pour un seul terme d'origine. Cependant, toutes les annotations proposées ne sont pas aussi pertinentes les unes que les autres par rapport au terme d'origine. Nous proposons donc une annotation sémantique floue des termes du tableau, en ordonnant par ordre de pertinence les termes de l'ontologie pour chacun des termes du tableau.

De même, lorsqu'un utilisateur interroge l'entrepôt sur les termes de l'ontologie, on veut que les réponses soient ordonnées par ordre d'adéquation de la réponse à la requête, donc par ordre de proximité du terme d'origine du tableau au terme de l'ontologie utilisé pour la requête.

Pour cela, nous représentons les annotations sous forme de données imprécises modélisées par des ensembles flous, où l'ensemble de référence est un ensemble discret : il s'agit de l'ensemble de tous les termes de l'ontologie. Le sous-ensemble flou représente la signification dans l'ontologie du terme d'origine, avec pour fonction d'appartenance une fonction qui associe à chaque terme de l'ontologie un degré de proximité avec le terme d'origine. Le classement des réponses pourra ainsi se faire suivant le degré de possibilité d'adéquation de la requête avec la donnée imprécise.

Les annotations sous formes d'ensembles flous peuvent être représentées dans le format XML en utilisant la formalisation suivante (Buche *et al.*, 2004) :
Soit A un ensemble flou défini sur un domaine de valeurs discret. A est représenté par un arbre de données ayant pour racine un nœud nommé *DFS* (Discrete Fuzzy Set) tel que, pour chaque élément x tel que $\mu_A(x) > 0$ (cf. définition 1), il existe un nœud fils nommé *valF*, qui ait deux nœuds fils respectivement nommés *Item* et *MD* (Membership Degree), ayant pour valeurs x et $\mu_A(x)$.

Exemple 3
La figure 3 est une représentation partielle du tableau de la figure 2 sous forme d'arbre

SML, en utilisant la représentation XML des ensembles flous à domaine de définition discret. Pour un attribut d'une relation, l'ensemble des balises finalVal *(figure 2) est remplacé par une seule balise* finalVal *avec un fils* DFS *qui a autant de sous-nœuds* valF *qu'il y a de termes de l'ontologie correspondant au terme d'origine. Les valeurs de* MD *présentées correspondent au score lexical de ressemblance présenté dans la partie 3 (voir détail des calculs dans le tableau 1).*

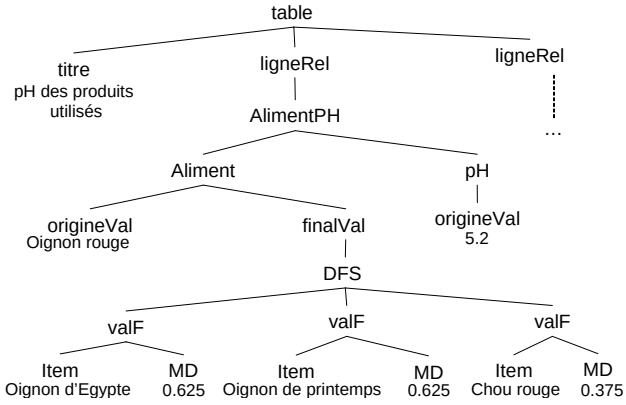


FIG. 3 – Arbre SML utilisant la représentation XML des ensembles flous à domaine de définition discret

3 Degré de proximité entre deux termes

Dans cette partie nous proposons une méthode pour annoter un tableau par les termes de l'ontologie. Dans un premier temps, nous considérons l'ensemble des termes des trois taxonomies, sans prendre en compte les relations de spécialisation entre ces termes. Nous présentons dans cette partie un calcul de degré de proximité entre un terme de l'ontologie (i.e. d'une des trois taxonomies) et un terme du tableau, appelé par la suite *terme extérieur à l'ontologie*.

3.1 Fonction de poids

L'annotation sémantique proposée par (Saïs *et al.*, 2005) se fonde sur une égalité de mots entre le terme extérieur à l'ontologie et les termes de l'ontologie (un terme est une suite de mots ayant un sens pour le domaine étudié, par exemple « oignon rouge » est un terme composé de deux mots, "oignon" et "rouge"). Tous les mots ont la même importance, et les différents termes de l'ontologie proposés pour représenter un terme extérieur à l'ontologie sont tous considérés sur le même plan, sans ordre de pertinence. Nous proposons dans ce papier de donner un ordre de pertinence aux termes de l'ontologie proposés pour l'annotation : pour cela nous calculons un degré de proximité entre le terme extérieur et les termes de l'ontologie. Le calcul de ce degré de proximité

repose sur la distinction entre des mots importants dans la sémantique d'un terme de l'ontologie, et des mots secondaires.
Chaque terme de l'ontologie est représenté sous la forme d'un ensemble de mots, où chaque mot est pondéré suivant son importance sémantique : poids de 1 pour un mot important dans la définition du terme, poids de 0 pour un mot « vide » (articles, conjonctions...) et poids intermédiaire pour un mot non vide mais d'importance mineure dans la sémantique du terme. La définition du poids des mots dans les termes de l'ontologie est réalisée par un expert du domaine, au même titre que la construction de l'ontologie elle-même : en effet le fait qu'un mot soit plus ou moins important dans un terme dépend étroitement de la signification du terme *pour le domaine*.
Les comparaisons de termes que nous utilisons sont fondées sur des égalités de mots, tirés de leur contexte et non ordonnés : nous n'avons notamment pas tenu compte des cas de négation, qui nécessiteraient des méthodes de traitement de la langue naturelle pour déterminer la portée de la négation.

Définition 3
On considère l'ensemble $T = \{C_1, ..., C_n\}$ des termes de l'ontologie. Chaque terme C_i est un ensemble de mots $C_i = \{m_{i,1}, m_{i,2}, ..., m_{i,k}\}$. On définit la fonction poids, de $\{(C, m) \mid C \in T$ et $m \in C\}$ dans $[0, 1]$, qui à un couple (C, m) associe l'importance du mot m dans le terme C, noté $poids(C, m)$.

Exemple 4
Considérons le terme de l'ontologie « lait de chèvre » : l'information principale est qu'il s'agit de lait ; cependant le fait que ce lait soit de chèvre et non de vache a également une certaine importance. Par contre, le mot "de" est un mot vide, nous considérons qu'il n'apporte pas d'information supplémentaire :
- $poids(lait\ de\ chèvre, lait) = 1$
- $poids(lait\ de\ chèvre, de) = 0$
- $poids(lait\ de\ chèvre, chèvre) = 0.8$

Pour chaque terme extérieur à l'ontologie que l'on veut rapprocher de l'ontologie, on étend la fonction *poids*. Cette extension est faite de façon automatique : l'importance sémantique des mots n'étant a priori pas connue pour les termes extérieurs à l'ontologie, on se contente de donner un poids de 0 aux mots vides et un poids de 1 à tous les autres mots. Nous avions envisagé d'utiliser des informations grammaticales pour l'attribution automatique de poids aux mots, par exemple un poids de 1 pour les noms et 0.5 pour les adjectifs. Cette solution n'a pas été retenue en raison de contre-exemples évidents (tels que « produit laitier » où l'adjectif "laitier" est le mot porteur de sens). Cependant une étude plus approfondie serait nécessaire pour déterminer si l'on peut tirer profit d'informations syntaxiques dans l'attribution des poids.

Définition 4
Soit $R = \{m_1, m_2, ..., m_k\}$ un terme extérieur à l'ontologie. On définit la fonction poids étendue à R, de $\{(C, m) \mid C \in T \cup \{R\}$ et $m \in C\}$ dans $[0, 1]$, qui à un couple (C, m) associe $poids(C, m)$ tel que :
- *Si C est un terme de l'ontologie, $poids(C, m)$ est tel que défini précédemment (définition 3)*

- *Si C est un terme extérieur à l'ontologie,*
  - *si m est un mot vide,* $poids(C, m) = 0$
  - *sinon* $poids(C, m) = 1$

Exemple 5
Considérons le terme « fromage de chèvre » rencontré dans un tableau extrait d'une publication scientifique. La fonction poids est étendue de la façon suivante :
- $poids(fromage\ de\ chèvre, fromage) = 1$
- $poids(fromage\ de\ chèvre, de) = 0$
- $poids(fromage\ de\ chèvre, chèvre) = 1$

Dans la suite, on appelle *fonction poids* la fonction poids étendue au terme extérieur (que l'on souhaite rapprocher de l'ontologie).

3.2 Score lexical de ressemblance

Le degré de proximité entre un terme extérieur à l'ontologie et un terme de l'ontologie que nous allons présenter maintenant est fondé sur des égalités de mots. Nous appelons ce degré *score lexical de ressemblance*.

Soit $R = \{m_1, m_2, ..., m_k\}$ un terme extérieur à l'ontologie, que l'on souhaite rapprocher de l'ontologie. Soit $C = \{m'_1, m'_2, ..., m'_j\}$ un terme de l'ontologie auquel on souhaite le comparer. Le calcul du score lexical de ressemblance entre R et C est donné par la formule suivante :

$$score_{lexical}(R, C) = \frac{\sum_{m,m' \in I} poids(R, m) + poids(C, m')}{\sum_{m \in R} poids(R, m) + \sum_{m' \in C} poids(C, m')} \qquad (1)$$

avec $I = \{(m, m') \mid m \in R, m' \in C$ et $lemmatisation(m) = lemmatisation(m')\}$

Plus les termes auront de mots en commun, et plus ces mots auront de l'importance sémantique dans le terme de l'ontologie, plus le score de ressemblance lexicale sera élevé. La lemmatisation, automatique, permet de comparer les mots indépendamment de leur forme grammaticale, et notamment de s'abstraire des formes plurielles : par exemple, *jus d'oranges* sera égal à *jus d'orange*.

Remarque 1
Si les termes sont égaux, le score lexical de ressemblance vaut 1. Si les termes n'ont aucun mot commun, le score lexical de ressemblance vaut 0.
Si tous les poids sont égaux à 1, le score lexical de ressemblance est égal au coefficient de similarité de Jaccard (Jaccard, 1912).

Exemple 6
Le tableau 1 montre le calcul du score lexical de resemblance entre le terme extérieur « oignon rouge » et les termes de l'ontologie « oignon d'Egypte », « oignon de printemps » et « chou rouge ». La distinction entre mots d'importance majeure ("oignon", "chou") et mots d'importance mineure ("Egypte", "printemps", "rouge") dans les trois termes de l'ontologie permet d'ordonner les annotations proposées, « oignon d'Egypte »

et « oignon de printemps » étant de meilleures annotations que « chou rouge ».
Par comparaison, si on supprime la notion de mot d'importance mineure (poids de 0.2 ramené à un poids de 1), alors le score lexical de ressemblance entre « oignon rouge » et chacun des trois termes de l'ontologie devient $\frac{1+1}{2+2} = 0.5$*. Les trois réponses « oignon d'Egypte », « oignon de printemps » et « chou rouge » sont alors présentées au même niveau dans l'annotation du terme « oignon rouge ».*

terme de l'ontologie	mots communs		mots différents		score	
	terme extérieur	terme de l'ontologie	terme extérieur	terme de l'ontologie	calcul	valeur
oignon d'Egypte	oignon, $poids = 1$	oignon, $poids = 1$	rouge, $poids = 1$	Egypte, $poids = 0.2$	$\frac{1+1}{2+1.2}$	0.625
oignon de printemps	oignon, $poids = 1$	oignon, $poids = 1$	rouge, $poids = 1$	printemps, $poids = 0.2$	$\frac{1+1}{2+1.2}$	0.625
chou rouge	rouge, $poids = 1$	rouge, $poids = 0.2$	oignon, $poids = 1$	chou, $poids = 1$	$\frac{1+0.2}{2+1.2}$	0.375

TAB. 1 – Calcul du score lexical de ressemblance du terme extérieur « oignon rouge » avec les termes de l'ontologie « oignon d'Egypte », « oignon de printemps » et « chou rouge »

3.3 Expérimentation

Pour évaluer notre approche, nous utiliserons la notion de *meilleur rapprochement*. Il s'agit du terme proposé dans l'annotation automatique qui est égal à celui proposé par l'expert dans une annotation manuelle. Nous nous intéresserons à l'existence ou non de ce meilleur rapprochement dans l'annotation automatique, ainsi qu'à l'ordre d'apparition (selon le score lexical de ressemblance) du meilleur rapprochement parmi les termes proposés dans l'annotation automatique.

3.3.1 Le jeu d'essai

Les noms d'aliments, en anglais, sont extraits de 12 tableaux contenus dans 12 publications en microbiologie alimentaire. On obtient une liste de 120 aliments différents à replacer dans l'ontologie. L'ontologie de référence est le Codex Alimentarius, répertoire anglophone de plus de 1600 aliments, utilisé par l'Organisation Mondiale de la Santé.

Les aliments ont été comparés manuellement à l'ontologie. Sur les 120 aliments initiaux, seuls 90 ont pu être replacés dans l'ontologie. En effet, le Codex Alimentarius traite surtout de matières premières, et certains aliments manufacturés ne trouvent pas leur place dans cette ontologie. Les résultats présentés portent donc sur ces 90 aliments.

Le Codex Alimentarius a été manuellement annoté, afin de donner les poids aux différents mots des termes de l'ontologie en fonction de leur importance sémantique. Pour faciliter l'annotation, nous avons décidé de ne conserver que trois niveaux de poids : 1 pour les mots porteurs de sens (au minimum un mot de poids 1 par terme de l'ontologie), 0 pour les mots vides et 0.2 pour les mots de moindre importance (le choix

d'un poids de 0.2 pour les mots d'importance mineure résulte d'essais préparatoires). Afin d'évaluer l'intérêt de la méthode de pondération des mots que nous proposons, les résultats avec cette pondération sont comparés avec ceux obtenus lorsque tous les mots non vides ont un poids de 1.

3.4 Les résultats

Sur les 90 aliments, 30 existent tels quels dans l'ontologie (à la lemmatisation près), ce qui se traduit par le meilleur rapprochement proposé en première position, avec un score lexical de ressemblance de 1. Sur les 60 aliments restants, le meilleur rapprochement fait partie des termes proposés dans l'annotation pour 40 aliments, dont 33 aliments pour lesquels le meilleur rapprochement est en première position (avec éventuellement des ex-aequo) par ordre de score lexical de ressemblance. On a donc 70% des aliments pour lesquels le meilleur rapprochement est donné en première position. Sur les 7 termes pour lesquels le meilleur rapprochement est trouvé, mais pas en première position, on trouve des faux-ami : « chicken scallop » (escalope de poulet) est rapproché de « scallop » (coquille St Jacques) prioritairement sur « chicken meat » ; on trouve également des généralisants : « cured meat (ham) » est rapproché de « meat » prioritairement sur « pig meat », car l'inclusion de mots donne un meilleur score que l'intersection de mots. Enfin, les 20 aliments pour lesquels on ne trouve pas le meilleur rapprochement n'ont aucun mot en commun avec ce meilleur rapprochement ; par contre, on peut souvent trouver des termes proches dans la hiérarchie. Par exemple, « fish filet » n'a pas pu être rapproché de « aquatic animal products » mais peut être annoté avec « dried fish », « marine fish » ou « freshwater fish ».

Lorsque le meilleur rapprochement est proposé dans l'annotation, il n'est pas toujours bien mis en valeur : il peut être présenté parmi d'autres termes ayant le même score. La figure 4 présente, pour les 40 termes pour lesquels le meilleur rapprochement est trouvé sans qu'il y ait égalité avec un terme de l'ontologie, le nombre de propositions concurrentes, c'est à dire le nombre de termes qu'il faudra présenter pour que le meilleur rapprochement soit dans la liste proposée.

On remarque que, pour un certain nombre de termes, le meilleur rapprochement a le meilleur score lexical de ressemblance sans proposition concurrente. Ces cas correspondent le plus souvent à des inclusions : l'ensemble de mots du terme de l'ontologie est inclus dans l'ensemble des mots du terme extérieur. Dans ces cas, le fait de différencier des mots de poids fort et des mots de poids faible dans les termes a moins d'impact. Pour les autres cas en revanche, le fait de distinguer les mots forts des mots faibles permet de discriminer les propositions : il y a moins de propositions concurrentes lorsque les mots peu importants se voient attribuer un poids de 0.2.

4 Prise en compte de la hiérarchie

Un second calcul de score de ressemblance permet de prendre en compte le fait que l'ontologie fournie est structurée suivant une relation de spécialisation. En effet, si un terme de l'ontologie subsume des termes représentatifs du terme extérieur à l'ontologie, il est probable que ce terme lui-même soit représentatif du terme extérieur à l'ontologie.

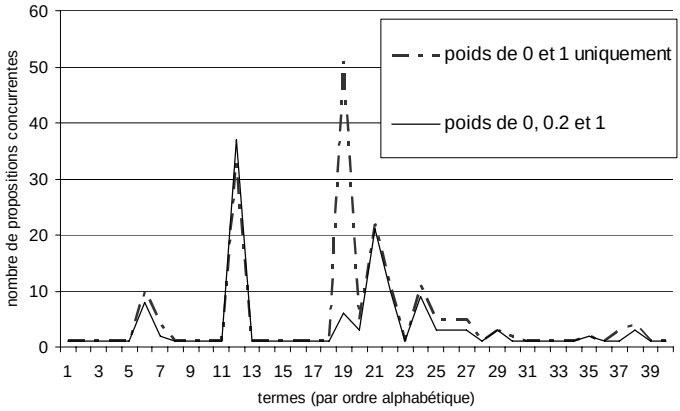


FIG. 4 – Nombre de propositions concurrentes, dans le cas de l'annotation de l'ontologie avec des poids de 0, 0.2 et 1 et dans le cas contrôle où tous les mots non vides ont un score de 1

4.1 Score hiérarchique de ressemblance

Le calcul du score hiérarchique de ressemblance d'un terme extérieur à l'ontologie avec un terme de l'ontologie se fonde sur le calcul du score lexical de ressemblance des nœuds-fils du terme de l'ontologie avec le terme extérieur.

Soient T l'ensemble des termes de l'ontologie, R un terme extérieur à l'ontologie et C un terme de l'ontologie. Soit F l'ensemble des fils directs de C par la relation de spécialisation, i.e. $F = \{c \in T \mid c \prec C \text{ et } \neg\exists c', c \prec c' \prec C\}$, avec la notation $c \prec C$ qui signifie c est *plus spécialisé* que C. Le score hiérarchique de ressemblance entre R et C est défini par :

$$score_{hierarchique}(R, C) = 1 - \prod_{f \in F} (1 - score_{lexical}(R, f)) \qquad (2)$$

En d'autres termes, plus il y a de fils ayant un score lexical de ressemblance non nul, et plus ces scores sont élevés, plus le score hiérarchique de ressemblance est élevé. On n'applique ce calcul qu'à un seul niveau (fils direct) afin d'éviter la sur-généralisation (savoir que le terme extérieur à l'ontologie peut être représenté par le terme universel ne nous intéresse pas).

Exemple 7
Considérons le terme « oignon rouge » étudié dans l'exemple 5. En le rapprochant des termes de l'ontologie, on trouve notamment les termes « oignon d'Egypte » et « oignon de printemps » ayant chacun un score lexical de ressemblance de 0.625 *avec « oignon rouge ». Ces deux termes ont un père commun, « végétaux à bulbe », dont le score lexical de ressemblance avec « oignon rouge » est égal à 0, puisqu'il n'y a aucun mot commun. On peut cependant calculer le score hiérarchique de ressemblance entre « oignon rouge » et « végétaux à bulbe » :* $1 - ((1 - 0.625) \times (1 - 0.625)) = 0.86$.

« Végétaux à bulbe » est le terme de l'ontologie le plus représentatif de « oignon rouge », les deux autres oignons proposés étant trop spécialisés : alors que le score lexical de ressemblance ne permettait pas de trouver le terme de l'ontologie le plus proche de « oignon rouge », le score hiérarchique de ressemblance place « végétaux à bulbes » en première position parmi les différentes propositions.

4.2 Expérimentation

Les 60 aliments que l'on peut manuellement placer dans l'ontologie mais qui ne sont pas égaux à un terme de l'ontologie (cf. section 3.4) sont utilisés pour cette expérimentation.

Le score hiérarchique de ressemblance permet de trouver le meilleur rapprochement pour 10 des 20 aliments pour lesquels le meilleur rapprochement avait un score lexical de ressemblance nul. Pour 9 de ces 10 aliments, le meilleur rapprochement est donné en première position par ordre de score hiérarchique de ressemblance.

L'utilisation du score hiérarchique de ressemblance entraîne une augmentation du nombre de termes proposés pour une annotation : en effet, pour chaque terme ayant un score lexical de ressemblance non nul, on remonte au généralisant. Ce généralisant peut être représentatif du terme extérieur à l'ontologie, et donc être une annotation intéressante, même si ce n'est pas forcément le terme le plus précis pour le représenter. Un classement binaire (soit le meilleur terme, soit un mauvais terme) n'a donc pas vraiment de sens. Pour évaluer nos résultats, on classe donc les termes de l'ontologie proposés pour un terme extérieur suivant trois catégories :

- terme le meilleur : le terme qui a été choisi dans le cadre d'une annotation manuelle ;
- terme acceptable : ce n'est pas le terme choisi dans le cadre de l'annotation manuelle, mais il représente tout de même bien le terme extérieur (généralisant ou fils peu éloigné sémantiquement). La notion de terme acceptable est liée à l'utilisation de l'annotation pour les requêtes : si le tableau constitue une bonne réponse pour une requête sur un terme de l'ontologie, alors ce terme est acceptable pour annoter le tableau ;
- rapprochement faux : faux-ami ou spécialisation trop éloignée.

On définit la précision comme étant le nombre de termes justes (le meilleur ou un terme acceptable) rapporté au nombre total de termes proposés dans l'annotation. On définit la couverture comme étant le nombre d'aliments pour lesquels on trouve le meilleur terme rapporté au nombre total d'aliments. Le score F1 est la moyenne géométrique de la précision et de la couverture.

Afin de limiter le nombre de termes proposés, pour chacun des 60 aliments, on ne conserve que les termes de l'ontologie pour lesquels au moins un des scores de ressemblance (lexical ou hiérarchique) est supérieur à un certain seuil. La figure 5 présente l'évolution de la précision, de la couverture et du score F1 en fonction du seuil choisi.

La valeur maximale du score F1 est obtenue pour un seuil de 0.6. Lorsqu'on choisit ce seuil, on a alors 37 aliments pour lesquels on trouve le terme le plus proche dans l'ontologie, 146 propositions de rapprochement réparties sur 37 aliments qui, bien que n'étant pas le meilleur rapprochement, sont jugées comme acceptables, et 114 propositions de

rapprochement fausses réparties sur 27 aliments. On obtient donc une couverture de 61% pour une précision de 62%.
A titre de comparaison, lorsqu'on utilise uniquement le score lexical de ressemblance avec le même seuil de 0.6, on obtient une couverture de 45% pour une précision de 71%.

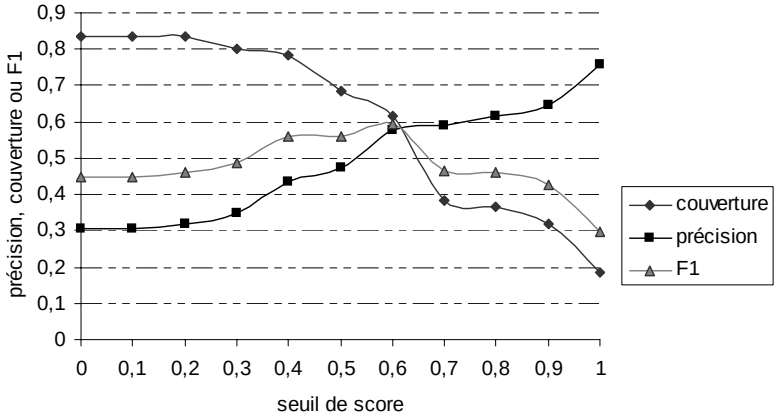


FIG. 5 – Précision, couverture et score F1 en fonction du seuil de score choisi

5 Conclusion et perspectives

Nous avons proposé une méthode d'annotation sémantique floue de termes présents dans des tableaux avec des termes d'une ontologie donnée. Cette annotation s'appuie sur un calcul de scores de ressemblance entre le terme à annoter et les termes de l'ontologie. Nous avons présenté un score lexical de ressemblance, qui utilise des comparaisons de mots : ce score intègre cependant des informations sémantiques via la pondération des mots dans les termes de l'ontologie. Ce calcul nous permet d'obtenir des annotations rangées par ordre de pertinence, mais ne permet pas de rapprocher des termes qui, bien que sémantiquement proches, n'ont aucun mot en commun. L'utilisation du score hiérarchique de ressemblance permet de remonter au généralisant, même si celui-ci n'a aucun mot en commun avec le terme du tableau. Cela permet d'augmenter la couverture, même si cela peut augmenter le bruit puisqu'on remonte également aux généralisants les scores des termes rapprochés par erreur.
Nous envisageons par la suite de nous intéresser à l'utilisation d'autres degrés de proximité, notamment en intégrant des opérateurs de mapping sémantiques fournissant un score numérique, tels que PANKOW (Cimiano *et al.*, 2004).

Dans la représentation des tableaux en SML (Saïs *et al.*, 2005), une heuristique est utilisée pour déterminer le type d'une colonne : elle s'appuie sur la reconnaissance d'une majorité des termes de la colonne comme appartenant à une catégorie de l'ontologie. Nous souhaitons étendre la notion d'annotation sémantique floue afin de représenter l'imprécision dans la détermination du type des colonnes, en tenant compte de l'imprécision dans la détermination de la catégorie de chacun des termes. Il existe

également une forme d'imprécision dans la reconnaissance des relations : imprécision sur le type des colonnes, relation représentée de façon partielle... Nous souhaitons étudier la façon dont nous pouvons représenter cette imprécision, en ne perdant pas de vue l'objectif d'interrogation des données.

La notion de rapprochement flou entre des termes avec une prise en compte de la hiérarchie est utilisée ici pour l'annotation de tableaux, mais elle pourrait également être étendue pour la mise en correspondance d'ontologies, où la notion de hiérarchie est présente dans l'ontologie connue mais aussi dans l'ontologie extérieure. Nous comptons développer cette approche dans le cadre de la mise en place d'une base de données sur les contaminants chimiques, qui intègre les données produites par différents instituts (DGAL, DGCCRF, INRA,...) ayant chacun leur propre référentiel.

Références

BUCHE P., DIBIE-BARTHÉLEMY J., HAEMMERLÉ O. & HOUHOU M. (2004). Towards flexible querying of XML imprecise data in a data warehouse opened on the web. In *Proceedings of the 6th International Conference On Flexible Query Answering Systems, FQAS 2004*, volume 3055 of *Lecture Notes in Artificial Intelligence*, p. 28–40, Lyon, France : Springer.

BUCHE P. & HAEMMERLÉ O. (2000). Towards contextual fuzzy querying of both structured and semi-structured imprecise data. In *Proceedings of the 4th International Conference on Flexible Query Answering Systems, FQAS 2000, Advances in Soft Computing*, p. 362–375, Warsaw, Poland : Physica Verlag.

BUCHE P., HAEMMERLÉ O. & THOMOPOULOS R. (2003). Integration of heterogeneous, imprecise and incomplete data : an application to the microbiological risk assessment. In *Proceedings of the 14th International Symposium on Methodologies for Intelligent Systems, ISMIS'2003*, volume 2871 of *Lecture Notes in Artificial Intelligence*, p. 98–107, Maebashi, Japan : Springer.

CIMIANO P., HANDSCHUH S. & STAAB S. (2004). Towards the self-annotating web. In *Proceedings of the 13th World Wide Web Conference, WWW04.*, p. 462–471, New York, United States : ACM.

E.DOT (2004). e.dot, revue intermédiaire du 29 juin 2004. Web site.
http ://www-rocq.inria.fr/verso/edot/.

GUARINO N. (1998). Formal ontology and information systems. In N. GUARINO, Ed., *Formal Ontology in Information Systems. Proceedings of FOIS'98*, p. 3–15, Trento, Italy : IOS Press. amended version : http ://www.loa-cnr.it/Papers/FOIS98.pdf.

JACCARD P. (1912). The distribution of flora in the alpine zone. *The New Phytologist*, **11(2)**, 37–50.

SAÏS F., GAGLIARDI H., HAEMMERLÉ O. & PERNELLE N. (2005). Enrichissement sémantique de documents sml représentant des tableaux. In *Actes des 5èmes journées Extraction et Gestion des Connaissances, EGC'2005*, p. 407–418, Paris, France.

SYM'PREVIUS (2004). Sym'previus, système de prévision du comportement des microorganismes dans les aliments. Web site. http ://www.symprevius.net.

ZADEH L. (1965). Fuzzy sets. *Information and Control*, **8**, 338–353.

ZADEH L. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, **1**, 3–28.

Modélisation et reconnaissance de situations dynamiques

Ali Aich − Sophie Loriette-Rougegrez

Département GSIT, Laboratoire ISTIT, FRE CNRS 27-32
Université de Technologie de Troyes (UTT),
{ali.aich, sophie.loriette}@utt.fr

Résumé : L'objectif de notre étude est de reconnaître l'occurrence de scenarii. Parmi les différentes approches utilisées, nous nous intéressons à la reconnaissance à base de modèles. Les travaux effectués sont présentés grâce à une application au projet **VAHM** (**V**éhicule **A**utonome pour **H**andicapés **M**oteurs) dont le but est de doter un fauteuil roulant d'une capacité d'anticipation du comportement souhaité par la personne handicapée. Il s'agit de reconnaître les nouveaux trajets effectués par l'utilisateur afin de lui proposer de futurs comportements. Après une présentation des différentes techniques de reconnaissance de scenarii et une description du projet VAHM, nous présentons le principe de création de classes de trajets, puis le modèle utilisé pour la reconnaissance. Ce modèle est un réseau de Petri qui permet d'une part de reconnaître les nouveaux trajets effectués et d'autre part, anticiper les futurs comportements de l'utilisateur. La partie expérimentation présente des résultats obtenus par le système implémenté.

Mots clés : Classification, Réseaux de Petri, Reconnaissance de scenarii, aide à la décision, raisonnement à partir de cas.

Thèmes de la communication : Apprentissage, Fouille et extraction (classification), raisonnement (aide à la décision, raisonnement à partir de cas).

1 Introduction

Notre étude s'inscrit dans le cadre de la supervision des systèmes dynamiques, fondée sur la reconnaissance de scenarii. Un scénario est une description d'une situation évolutive. Il prend la forme dans notre travail d'un ensemble d'états associés par des contraintes temporelles (Dousson, 1994). Cet article présente l'utilisation de la reconnaissance de scenarii dans le cadre du projet ***VAHM*** (**V**éhicule **A**utonome pour **H**andicapés **M**oteurs). L'objectif du projet est de doter un fauteuil roulant d'une capacité d'anticipation de la trajectoire souhaitée par la personne handicapée, lui évitant de répéter les mêmes commandes de directions lors du suivi d'une trajectoire déjà empruntée.

La structuration de l'article est la suivante. Dans la deuxième section, nous présentons le projet VAHM, puis nous introduisons notre problématique. Dans la troisième section, nous présentons différentes approches de reconnaissance de scenarii. Ensuite, dans la quatrième section, nous décrirons l'approche proposée en tenant compte de la problématique posée, suivie par des expérimentations effectuées sur le système développé. La dernière section conclut le présent article.

2 Présentation du projet VAHM

Le Véhicule Autonome pour Handicapés Moteur (***VAHM***) se place dans le cadre de la robotique mobile. Il a pour rôle essentiel de se déplacer dans un environnement plus ou moins connu et plus au moins variable dans le temps. Dans le cadre de la conduite d'un fauteuil par un utilisateur handicapé, il est nécessaire que celui-ci soit maître de son « véhicule » afin de ne pas se sentir transporté, mais être acteur du mouvement (Morere, 2002).

La définition du mouvement à imposer au fauteuil est réalisée à l'aide d'une interface (joystick ou système de balayage) qui traduit l'action à générer en terme de direction. Le fauteuil prend en charge l'évaluation de cette information et la traduit en comportement à effectuer. La planification du mouvement global est déterminée entièrement par l'utilisateur, par contre il est envisagé que le fauteuil vienne assister la personne dans la reproduction de chemins fréquemment employés. Si à la sortie d'une pièce particulière, l'utilisateur a l'habitude de suivre une trajectoire, il serait intéressant que le système reproduise cette trajectoire en évitant à l'utilisateur de reproduire les différentes commandes de direction nécessaires.

Afin de clarifier la description du projet présenté, il est important de préciser la signification de certains termes utilisés (Pruski & Ennaji*.,* 2001) :

- **Situation :** ensemble des informations disponibles sur l'état d'un système à un instant donné.
- **Comportement :** mouvement élémentaire du fauteuil, obtenu par la traduction des données issues des capteurs. Dans le but de faciliter la compréhension des exemples, chaque comportement sera associé à un identificateur[1].
- **Trajet :** un trajet correspond à un déplacement du fauteuil. Il est constitué d'une suite de comportements élémentaires qui s'enchaînent logiquement, chacun de ces comportements peut se produire un nombre quelconque de fois. Le trajet peut donc comporter par exemple

[1] Neuf comportements sont définis : Arrêt (**1**), Evitement d'obstacles (**2**), retour arrière à droite (**3**), retour arrière à gauche (**4**), Suivi de direction (**5**), Suivi de mur droit (**6**), Suivi de mur gauche (**7**), Backtracking (**8**), Suivi d'espace libre (**9**).

6 *retours arrière à droite* puis 2 *évitements d'obstacles* puis 3 *retours arrière à gauche.*

- **Trajectoire :** une trajectoire est un ensemble de trajets ayant les mêmes points de départ et d'arrivée.

Nous avons utilisé les données acquises lors de l'exécution répétée de trajectoires. En effet, notre système doit être capable de reconnaître une trajectoire même si elle diffère un peu de celles déjà effectuées.

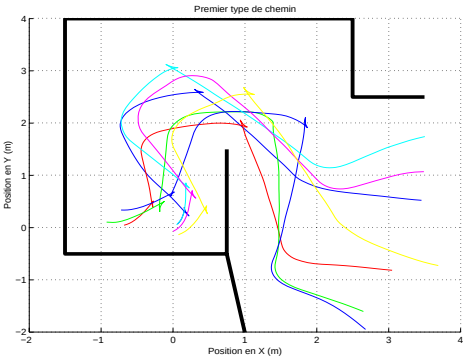


Fig. 1 – Différents trajets effectués pour la même trajectoire

L'objectif de ce projet est de mettre au point une méthode capable, à partir des comportements réalisés, de reconnaître la trajectoire effectuée par le fauteuil afin de proposer des futurs comportements à l'utilisateur. Ceci nécessite d'utiliser un ou des modèles séquentiels de représentation des situations, et donc de prendre en compte l'évolution du mouvement ; et aussi la prise en compte de la similarité des trajectoires ayant mêmes points de départ et d'arrivée.

3 Reconnaissance de scenarii

Cette section décrit différentes techniques de reconnaissance de scénarii. Il existe différentes approches pour la reconnaissance de scenarii. En général, la reconnaissance repose sur deux types d'approches principales : la reconnaissance d'une situation sur la base de sa ressemblance totale ou partielle avec une situation déjà reconnue, sur lequel repose le *raisonnement à partir de cas* (RàPC) (Kolodner, 1993), ou la reconnaissance mettant en oeuvre un ou plusieurs modèles décrivant l'évolution envisageable des situations tels que *les réseaux de Petri (RdP)* (Rene & Alla., 1992) ou *les chroniques* (Ghallab & Dousson, 1994).

Le raisonnement à partir de cas est une approche de résolution de problèmes basée sur la réutilisation par analogie d'expériences passées appelées « *cas source* » au cours d'un cycle de raisonnement, pour résoudre un nouveau problème appelé « *cas cible* ». Le traitement est alors effectué en quatre phases, étant identifié le nouveau problème à résoudre (Loriette, 1998) :

- Phase de *recherche* pour identifier le ou les cas ressemblant le plus au cas cible, les meilleurs cas sont alors retournés.
- Phase de *Réutilisation* qui opère la fusion des cas déjà identifiés dans la phase précédente pour construire une solution au cas cible.
- Phase *d'adaptation* au cas cible des solutions trouvées.
- Phase d'*apprentissage* qui permet d'ajouter le cas cible résolu à la base de cas pour une éventuelle réutilisation.

Cette approche est adéquate pour les domaines où *la similarité* entre les descriptions de problèmes nous donne une indication.

Le deuxième type d'approche mis en avant ci-dessus repose en grande partie sur des modèles de type réseaux de Petri. Les réseaux de Petri sont utilisés pour la modélisation des systèmes dynamiques (Rene & Alla*.,* 1992). Un RdP est un graphe bipartie, composé de places indiquant les différents états du système modélisé, et de transitions déterminant les conditions de changement d'états. Parfois, on peut indiquer la durée d'un état du système on associant des contraintes temporelles aux places (*RdP p-temporisé*) ou aux transitions (*RdP t-temporisé*).

Le marquage d'une place représenté par la présence d'un jeton dans cette place, désigne le fait que le système est dans l'état indiqué. Ce type de modèle décrit le comportement normal du système qui peut être comparé avec le comportement en cours pour la phase de reconnaissance.

L'approche proposée par Dousson et Ghallab utilise les *chroniques* dans la reconnaissance de scenarii : elle est fondée sur un graphe temporel représentant les contraintes entre les différents évènements (relation de précédence temporelle) et la phase de reconnaissance consiste en une propagation de contraintes sur ce graphe (Ghallab & Dousson, 1994).

Différentes approches de reconnaissance de scenarii ont été présentées. Les choix que nous avons faits se situent pleinement dans les conditions évoquées précédemment et ils réalisent un compromis entre les approches précédemment présentées. En effet, nous avons tiré partie des avantages du RàPC pour traiter la similarité entre trajets de la même trajectoire, ensuite établir un modèle de reconnaissance à base de réseau de Petri. Ce type de modèles a été choisi car ils sont bien adaptés pour décrire la dynamique de tels systèmes (description de passage d'un état à un autre).

4 Approche envisagée

Un certain nombre de trajets a été effectué par le fauteuil pour une même trajectoire (figure 1). Chacun d'eux est une succession logique de comportements. L'objectif est d'établir un modèle de reconnaissance des nouveaux trajets effectués afin d'anticiper le futur comportement de l'utilisateur. Notre approche est basée sur deux étapes importantes en commençant par la classification des trajets *similaires* (trajets de la même trajectoire), et ensuite la modélisation de chaque représentant de classe par un réseau de Petri.

4.1 Constitution des classes de trajets

Plusieurs trajets ont été effectués pour la même trajectoire. Pour une meilleure utilisation et afin d'éviter de stocker des trajets similaires, un modèle englobant tous les trajets effectués pour une trajectoire sera établi. Il sera utilisé par la suite pour déterminer l'appartenance d'un nouveau trajet effectué à cette classe. Nous décrivons maintenant la construction des classes de trajets.

Les comportements d'un trajet peuvent apparaître plusieurs fois consécutivement. Nous proposons donc de les mémoriser dans un tableau de deux lignes : *la première ligne représente les comportements qui le constituent (**comp_trajet**) et la deuxième le nombre d'occurrences de chacun (**occ_comp**)*. Ce type de mémorisation est choisi pour faciliter le parcours des trajets stockés et faire correspondre chaque comportement à son nombre d'occurrences (figure 2).

Une fois que ce travail a été effectué, les trajets de la même trajectoire sont regroupés dans la même classe. Le principe consiste à comparer les séquences de comportements constituant les différents trajets à classifier (***comp_trajet***) en recherchant les transformations (*éditions*) nécessaires pour passer d'une séquence à une autre. Les transformations nous permettent par la suite de calculer une distance d'édition entre deux trajets.

- **Définition 01** : une *édition* utilisée dans la comparaison des chaînes de caractère (Ganascia, 2001) est une transformation élémentaire qui remplace un caractère dans une chaîne par un autre éventuellement vide. Elle peut être une *insertion* (Ins), une *suppression* (Sup) ou une *substitution* (Sub).
- **Définition 02 :** une *distance d'édition* entre deux séquences correspond à la somme des coûts d'éditions qui transforment une séquence en une autre. Dans notre cas, on a associé le coût **1** à l'insertion, la suppression et la substitution d'un caractère.

Le résultat de la mise en correspondance des séquences est appelé *un alignement* (Varre, 2000). Il est obtenu en ajoutant des espaces (notés /) dans les séquences tel que toutes les séquences ont la même longueur (figure 2). Ce traitement permet de calculer le meilleur représentant d'une classe de trajets.

4.1.1 Alignement des séquences de trajets

Plusieurs algorithmes appelés algorithmes de *programmation dynamique* sont utilisés pour aligner les séquences en trois phases (Duchesne, 2004) : aligner les séquences deux à deux, rechercher la séquence qui minimise les distances d'éditions appelée *séquence référence*, ensuite aligner toutes les autres séquences séquentiellement par rapport à cette dernière. Ce type d'algorithme est utilisé essentiellement en bioinformatique pour comparer les séquences d'ADN (Varre, 2000).

Pour l'alignement des séquences deux à deux, nous considérons deux séquences x, y données sous forme de deux tableaux de taille respective n et m, notés x[1..n] et y[1..m]. Il est alors possible de construire une matrice de dimension (n+1)×(m+1) appelée « *matrice des distances d'édition* (EDIT) » dont ses éléments sont calculés par la formule suivante :

$$\text{edit}(i,j) = \min \begin{pmatrix} \text{edit}(i-1,j-1) + \text{Sub}(x[i],y[j]) \\ \text{edit}(i,j-1) + \text{Ins}(y[j]) \\ \text{edit}(i-1,j) + \text{Sup}(x[i]). \end{pmatrix} \quad (1)$$

Pour calculer l'alignement entre les deux séquences considérées, il suffit de tracer un chemin dans la matrice EDIT allant de la case edit[n+1,m+1] jusqu'à la case edit[1,1] et en passant d'une case à celle qui nous a permis de la calculer (Charras & Lecroq, 1998).

L'exemple de la figure 2 montre l'alignement de trois trajets mémorisés respectivement dans les tableaux : *trajet01, trajet02, trajet03.*

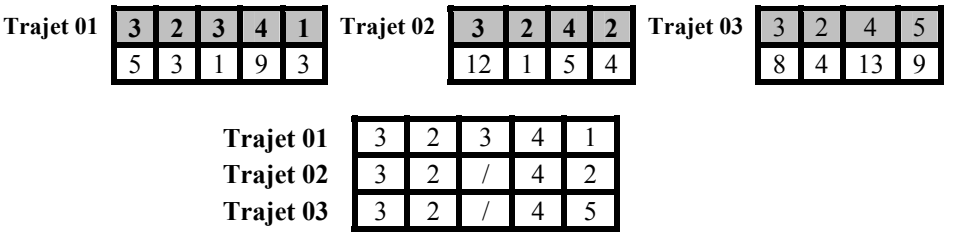


Fig. 2 – Alignement multiple des trajets

4.1.2 Calcul des représentants des classes de trajets

Après l'alignement des différentes séquences de trajets appartenant à la même classe, l'étape suivante consiste à déterminer son représentant. Ce dernier est une matrice de trois lignes : la première ligne représente les différents comportements élémentaires (si à une colonne de l'alignement multiple, tous les trajets ont le même comportement, alors ce dernier est ajouté à la même position dans le représentant de la classe sinon un autre comportement est inséré appelé comportement facultatif *F*), le minimum des occurrences des comportements est inséré à la deuxième ligne, et le maximum à la troisième ligne.

Par cet algorithme, le représentant des trois trajets de la figure 2 est calculé. Par exemple, les trois trajets commencent par le comportement ***3*** réalisé 5 fois dans le trajet01, 12 fois dans le trajet02 et 8 fois dans le trajet03. Dans ce cas, à la même position du représentant le comportement ***3*** est inséré dans la première ligne, 5 (minimum entre 5, 8, 12) dans la deuxième ligne et 12 (maximum entre 5, 8, 12) dans la troisième ligne.

3	**2**	**3**	**4**	**F**
5	1	0	5	3
12	4	1	13	9

Fig. 3 – Calcul des représentants de classes

4.2 Modélisation des représentants de classes

Un représentant d'une classe de trajets est une succession de comportements. Chacun d'eux doit être réalisé n fois (n est compris entre le nombre d'occurrences minimal et le nombre d'occurrences maximal qui lui sont associés) avant de passer au suivant. Le formalisme que nous avons choisi est celui des réseaux de Petri (Valette, 2000) car ce type de modèle est bien adapté pour décrire la dynamique de tels systèmes (description du passage d'un état à un autre). La figure 4 représente le réseau de Petri modélisant le représentant de la classe de trajets de la figure 3.

4.2.1 Places et transitions

Les places d'un réseau de Petri représentent les différents états du système modélisé. Dans notre cas, un état correspond à un comportement en cours de réalisation. Par conséquent, chaque comportement constituant un représentant de classe sera modélisé par une place. Les nombres d'occurrences de chacun sont associés à sa place correspondante faisant référence à la représentation des contraintes temporelles dans les modèles de chroniques (Boufaied, 2002).

Par exemple, le représentant de classe de la figure 3 commence par le comportement ***3*** réalisé entre 5 et 12 fois. Alors ce dernier est représenté dans le RdP correspondant par une place associée à ce comportement et l'intervalle [5, 12] (figure 4).

Les transitions d'un réseau de Petri indiquent les conditions de changement d'états (conditions de passage d'un comportement à un autre). Le fauteuil doit réaliser au moins le nombre minimal d'occurrences du comportement en cours avant de passer au suivant. Dans le modèle de la figure 4, le fauteuil doit réaliser le premier comportement (***3***) au moins 5 fois avant de passer au suivant (***2***).

4.2.2 Jetons

Dans un réseau de Petri, un « *conflit structurel* » correspond à l'existence d'une transition avec au moins deux places en amont (Rene & Alla, 1992). Dans notre cas, une transition est en conflit structurel si elle a plus d'une place en amont d'elle, représentant un comportement associé à un nombre minimal d'occurrences nul. Par exemple, la troisième transition du modèle de la figure 4 est en conflit structurel.

La distribution des jetons dans le modèle dépend des types de conflits qui peuvent se produire. Par exemple, un jeton est associé à la troisième place du modèle pour que la première et la troisième transitions puissent être franchies dans cet ordre en cas de reconnaissance d'un trajet constitué successivement des trois comportements ***3, 2, 4***.

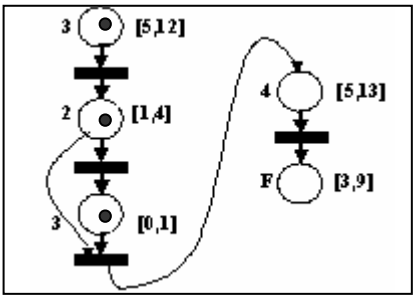


Fig. 4 – Modèle d'un représentant de classe

A ce niveau, chaque représentant de classe est modélisé par un réseau de Petri. En exploitant l'une des caractéristiques importantes de ce type de modèles qui est la *synchronisation*, on a choisi de les regrouper dans un modèle global commençant par une place appelée *début modèle* et terminant par une autre appelée *fin modèle*. La première place est suivie par une transition regroupant toutes les premières places des modèles, et la deuxième est précédée par une transition qui a toutes les dernières places des modèles regroupés en amont d'elle. Ce type de regroupement des modèles permet un *gain de temps* dans la phase de reconnaissance car les différents modèles des représentants de classes sont parcourus en même temps.

La figure 5 présente un exemple de regroupement des modèles de deux représentants de classes. Le premier modèle constitué des trois places ***P1M1, P2M1, P3M1*** associées respectivement aux comportements ***3, 2, 3*** ; et le deuxième modèle constitué des places ***P1M2, P2M2, P3M2*** associées aux comportements **2, *4, 5***.

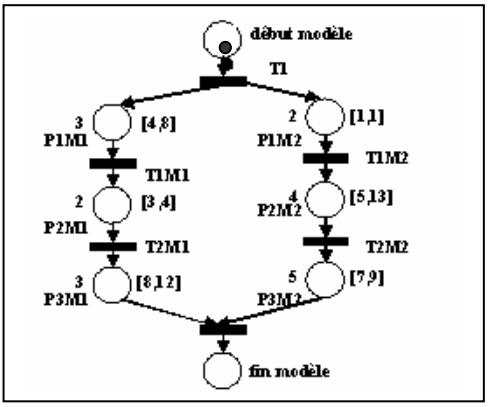


Fig. 5 – Regroupement des modèles

5 Reconnaissance et prédiction

La reconnaissance a pour but de déterminer l'appartenance d'un nouveau trajet effectué (mémorisé sous forme d'un tableau de deux lignes, figure 2) aux classes modélisées précédemment. Cette tâche se base sur le parcours du réseau de Petri regroupant les représentants de classes, et le tableau mémorisant le trajet à reconnaître en parallèle. Initialement, on commence par la transition en aval de la première place (place *début modèle, T1*) et du premier comportement.

Le franchissement d'une transition validée (il existe une (des) place (s) *PR* (s) en aval d'elle, associée (s) au comportement considéré et un intervalle vérifiant son nombre d'occurrences) implique le passage au comportement suivant et à la (aux) transition(s) en aval de la (des) place (s) *PR* (s). La reconnaissance se termine si l'une des ces trois conditions est vérifiée : le dernier comportement du trajet est atteint, la fin du modèle (la transition en amont de la place *fin modèle* est atteinte) ou il n' y a aucune transition validée à franchir.

La phase de prédiction est une conséquence de la reconnaissance. Elle consiste à proposer à l'utilisateur le (les) comportement(s) associé(s) à la (aux) place(s) en aval de la (des) transition(s) en cours (transitions de fin de reconnaissance).

Le tableau de la figure 6 présente la reconnaissance d'un trajet constitué des deux comportements 3, 2 répétés successivement 5 et 4 fois (333332222) par le modèle de la figure 5. La prédiction consiste à proposer à l'utilisateur le comportement associé à la place en aval de la dernière transition de reconnaissance (le comportement 3 associé à la place P3M1 qui pourra être répété entre 8 et 12 fois).

Places des jetons	**Comportement considéré**	**Transition à franchir**
Place début modèle *place initiale*	3 *comportement initial*	T1
P1M1	2	T1M1
P2M1 *fin de reconnaissance*	- *fin du trajet*	-

Fig. 6 – Exemple de reconnaissance d'un trajet

6 Expérimentation et résultats

Afin de valider le modèle et les principes définis, un logiciel est développé en Delphi VII afin de mettre en œuvre les différents algorithmes proposés. Il permet de visualiser l'ensemble des traitements effectués (classification des trajets, reconnaissance et prédiction).

Ce logiciel récupère en entrée des trajets représentés sous forme de séquences de comportements. Les algorithmes de classification et de reconnaissance implémentés à partir des principes décrits précédemment permettent de classifier, reconnaître des trajets effectués par le fauteuil expérimental, afin de proposer des futurs comportements à l'utilisateur.

Pour la classification, les trajets à mettre dans la même classe sont récupérés en entrée sous forme de fichiers textes, mémorisés en structures de tableaux ensuite alignés pour constituer le représentant de la classe suivant les algorithmes décrits précédemment. Quatre représentants de classes ont été calculés. L'expérience présentée consiste à calculer le représentant de quatre trajets après leur mémorisation (figure 7).

La figure 8 présente la reconnaissance d'un nouveau trajet récupéré par le système sous forme d'un fichier texte. Après sa mémorisation, le modèle regroupant les quatre représentants de classes est parcouru pour déterminer son appartenance à l'une des classes, ensuite prédire le futur comportement du fauteuil.

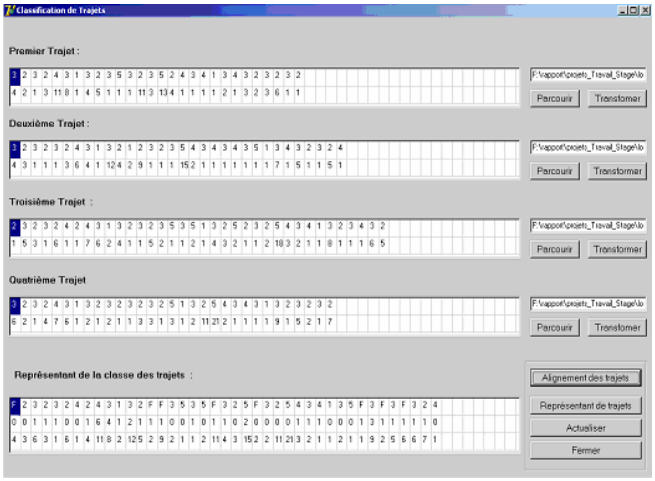


Fig. 7 – Exemple de classification de trajets

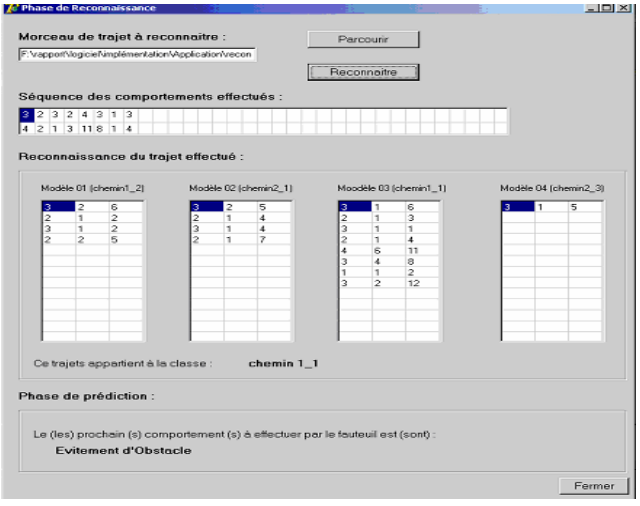


Fig. 8 – Exemple de reconnaissance d'un nouveau trajet

7 Conclusion et perspectives

A partir des particularités du projet *VAHM* présenté précédemment, des techniques empruntées au raisonnement à partir de cas et aux réseaux de Petri, nous avons défini une approche hybride combinant les avantages du RàPC permettant la

classification, la reconnaissance des trajets effectués par le fauteuil afin d'anticiper ses futurs comportements. Les résultats auxquels a abouti ce travail sont les suivants :

- L'établissement *a priori* d'un modèle de reconnaissance en deux étapes : la classification des trajets similaires, puis la modélisation de chaque représentant de classe par un réseau de Petri. Quatre classes de trajets ont été proposées.
- L'utilisation du modèle établi précédemment pour la reconnaissance hors ligne des nouveaux trajets effectués par le fauteuil.
- La prédiction des futurs comportements du fauteuil en se basant sur les résultats de la reconnaissance.

Une des limites de l'approche proposée dans cet article est la difficulté d'établissement du modèle de reconnaissance a priori. Il est généralement difficile d'établir des modèles de reconnaissance des situations dynamiques a priori. La raison est que la grande variabilité de ces situations rend leurs appariements avec le modèle difficile.

Nous envisageons de proposer une nouvelle approche permettant la reconnaissance et l'apprentissage de nouvelles situations en ligne. L'objectif est de construire progressivement le ou les modèles de reconnaissance, en se basant sur les situations déjà reconnues. L'idée est assez proche des systèmes de classification automatique ou de généralisation, mais la particularité repose sur la considération de situations dynamiques basées sur une représentation continue et spatio-temporelle.

Références

BOUFAIED A (2002). Modélisation de chroniques par réseaux de Petri pour la détection distribuée de défaillances du procédé. *3ème congrès des doctorants de l'Ecole Doctorale SYSTEMES.* LAAS-CNRS.

CHARRAS C. & LECROQ T. (1998). Sequence comparison. *ABISS(Atelier Biologie Informatique Statistique Socio-linguistique).* University of Rouen.

DOUSSON C. (1994). Suivi d'évolution et reconnaissance de chroniques. *Thèse de doctorat.* Université de Paul Sabatier de Toulouse.

DUCHESNE J-E. (2004). Programmation dynamique : alignement multiple. *Cours de bioinformatique.* Université de Montréal.

GANASCIA J-G. (2001). Extraction automatique de motifs syntaxiques. *TALN*. Tours.

GHALLAB M. & Dousson C. (1994). Suivi et reconnaissance de chroniques. *Revue d'intelligence artificielle*, vol 8.

KOLODNER J. (1993). Case-based reasoning. San Mateo, CA : Morgan Kaufman.

LORIETTE S. (1998). Raisonnement à partir de cas pour des évolutions spatio-temporelles de processus. *Revue internationale de géomatique, journées Cassini*, vol. 8.

MORERE Y. (2002). Reconnaissance de trajet en environnement fermé. *Compte-rendu d'une réunion*.

PRUSKI A. & ENNAJI M. (2001). Symbiotic Human-Machine Architecture for a Smart Wheelchair Control. *European Conference for the Advancement of Assistive Technology in Europe*.

RENE D. & ALLA H. (1992). Du Grafcet aux réseaux de Petri. Edition Hermes, pp 19-164. Paris.

VALETTE R. (2000). Les réseaux de Petri. *Support de cours ENAC*. LAAS, Toulouse.

VARRE J-S. (2000). Alignement multiple et applications. *Support de cours*. LIFL, Lille.

Construction de fonctions de décision performantes et de complexités réduites avec des SVM

G. Lebrun[1], C. Charrier[1], O. Lezoray[1], H. Cardot[2]

[1] LUSAC EA 2607, groupe Vision et Analyse d'Image, IUT Dept. SRC, 120 Rue de l'exode, Saint-Lô, F-50000, France
{gilles.lebrun, c.charrier, o.Lezoray}@chbg.unicaen.fr

[2] Laboratoire Informatique (EA 2101), Université François-Rabelais de Tours, 64 Avenue Jean Portalis, Tours, F-37200, France
hubert.cardot@univ-tours.fr

Résumé :
La puissance actuelle des ordinateurs permet de collecter des masses de données décrites par un nombre d'attributs de plus en plus grand. Les méthodes d'extraction des connaissances à partir des données (ECD) sont devenues indispensables pour la conception de systèmes d'information efficaces. Les méthodes d'apprentissage sont donc très utilisées en ECD avec comme objectif de produire par induction une fonction de décision ayant les meilleures performances possibles en généralisation. Les SVM sont des classificateurs possédant de très bonnes performances théorique et pratique, mais leur temps d'apprentissage devient rapidement prohibitif lorsque la taille de la base d'apprentissage augmente. Nous proposons d'exploiter la recherche tabou pour le réglage des hyperparamètres et la sélection des attributs pertinents. Cette heuristique est combinée à une quantification vectorielle afin d'obtenir une représentation réduite de la base d'apprentissage. Les expérimentations réalisées montrent que cette nouvelle technique permet de produire avec des SVM, dans un laps de temps réduit, des fonctions de décision binaires de complexités réduites qui ont de très bonnes performances en généralisation.

Mots-clés : Classification, SVM, ECD, Apprentissage, Recherche Tabou, Quantification Vectorielle, Sélection d'attributs

1 Introduction

Les systèmes informatiques actuels permettent d'obtenir des bases de données comportant un grand nombre d'instances (ou d'exemples), chaque instance étant décrite par une multitude d'attributs. L'augmentation de la taille de la base s'accompagne généralement de redondances parmi les instances et d'attri-

buts plus ou moins corrélés ou non significatifs. L'extraction de connaissances à partir de cette masse de données par des méthodes de classification est alors un problème difficile. En effet, l'objectif est d'induire une fonction de décision ayant de très bonnes performances en généralisation. Le choix des SVM s'impose car ce sont des classificateurs très performant en généralisation pour des raisons théoriques (Vapnik, 1998). Les temps de calcul pour produire la ou les fonctions de décision avec les SVM deviennent rapidement prohibitifs (Loosli *et al.*, 2004) avec l'augmentation du nombre d'exemples (Platt, 1999) (et dans une moindre mesure avec le nombre d'attributs). De plus, dans le cas d'une exploitation en temps réel des fonctions de décision, il est important que la sélection d'une classe soit rapide. Par conséquent, le nombre d'attributs à calculer pour caractériser un objet doit être le plus faible possible. Il est aussi important que la complexité intrinsèque de la fonction de décision soit réduite. Dans le cas des SVM, cela correspond à réduire le nombre de vecteurs de support (cf. section 2).
Nous proposons une nouvelle méthode d'apprentissage des SVM qui prend en considération les remarques précédentes. Elle se place dans le cadre général de la recherche d'un modèle optimal permettant d'induire une fonction de décision performante en généralisation et de complexité réduite. La recherche du modèle optimal correspond à la détermination des valeurs des hyperparamètres des SVM et à la sélection du meilleur sous-ensemble d'attributs (Chapelle *et al.*, 2002). Comme la sélection du modèle doit être réalisée à partir d'un espace de dimension très importante et pour éviter tout problème d'explosion combinatoire, une heuristique sous-optimale a été choisie. Parmi ces heuristiques la recherche tabou a été sélectionnée pour ses très bonnes performances, en particulier sur des problèmes de nature similaire (Cawley, 2001; Korycinski *et al.*, 2004).
L'idée principale de notre méthode est d'utiliser la quantification vectorielle pour produire des versions simplifiées de la base initiale. Ces versions simplifiées permettent d'estimer rapidement le pouvoir généralisateur d'un modèle (Lebrun *et al.*, 2004). Elles permettent aussi de réduire la complexité des fonctions de décision produites. L'autre apport important de notre méthode réside sur le fait que le choix du niveau de simplification est réalisé par la recherche tabou.
L'évalution de la performance en généralisation est un problème délicat et doit être correctement mené afin d'éviter tous les problèmes de sur-apprentissage (Joachims, 2000). Dans les expérimentations réalisées, nous avons utilisé deux méthodes d'estimation de l'erreur de généralisation pour la sélection du modèle optimal. Dans les deux cas, cette estimation a été pénalisée en fonction de la complexité de la fonction de décision produite pour favoriser les solutions de complexité moindre.

2 Machines à Vecteurs de Support

La théorie de l'apprentissage statistique de Vapnik et de Chervonenkis (Vapnik, 1998) a conduit au développement d'une classe d'algorithmes connue sous le nom de Support Vector Machines (SVM) qui sera traduite par machine à vec-

teurs de support. Une des originalités de la méthode est de produire une fonction de décision binaire qui n'utilise qu'un sous-ensemble de la base d'apprentissage. Les éléments de ce sous-ensemble sont nommés Vecteurs de Support (VS). Soit une base d'apprentissage $S_a = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ composé de m couples (vecteur d'attributs, classe) avec $x_i \in \mathcal{R}^n$ et $y_i \in \{-1, +1\}$. L'algorithme des SVM projette les vecteurs x_i dans un espace de travail $\mathbf{H}$ à partir d'une fonction non linéaire $\phi : \mathcal{R}^n \rightarrow \mathbf{H}$. L'hyperplan optimal de séparation des deux classes dans l'espace $\mathbf{H}$ est ensuite recherché. Cet hyperplan $(\mathbf{w}, b)$ matérialise la frontière de séparation entre les deux classes. La classe y d'un nouvel exemple $\mathbf{x}$ est définie par : $y = sign(\mathbf{w} \cdot \phi(\mathbf{x}) + b)$. L'hyperplan est optimal s'il maximise la distance qui le sépare des exemples dont il est le plus proche. Cette distance est usuellement appelée marge du classificateur. Il a été démontré (Vapnik, 1998) que maximiser cette marge correspond à maximiser le «pouvoir» généralisateur du classificateur. Vapnik (Vapnik, 1998) a proposé une version duale du problème qui revient à minimiser $\mathcal{W}(\alpha) = \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{m} \alpha_i$ sous les contraintes $\forall i \in [1, \ldots, m] : \sum_{i=1}^{m} y_i \alpha_i = 0\,,\ 0 \leq \alpha_i \leq C$. L'avantage de cette formulation est qu'elle utilise une fonction noyau K respectant l'égalité $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. Ce qui évite de travailler directement dans l'espace de projection $\mathbf{H}$. La solution optimale α^* définit l'ensemble V_S des vecteurs de support ($\forall i \in [1, \ldots, m], i \in V_S : \alpha_i > 0$) et permet de définir la fonction de décision (Vapnik, 1998) :

$$f(\mathbf{x}) = sign\left(\sum_{i \in V_S} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}) + b^*\right) \tag{1}$$

Le compromis entre erreur de classification et complexité du modèle dépend du paramètre C. Un algorithme efficace SMO (Platt, 1999) a été proposé par Platt pour résoudre le problème dual. Les SVM étant des classificateurs binaires, la résolution d'un problème multiclasses est réalisée en le transformant en une combinaison de problèmes binaires (Hsu & Lin, 2002), ceci restant cependant un domaine de recherche très ouvert.

3 Méthode Tabou

La Recherche Tabou (RT) est une métaheuristique pour l'optimisation difficile qui s'est développée dans les années 80 (Glover, 1989a,b). Elle s'est révélée particulièrement efficace et a été appliquée avec succès à de nombreux problèmes (Hao *et al.*, 1999). Elle est basée sur une recherche itérative qui choisit dans un voisinage restreint la meilleure solution même si elle est plus mauvaise que celle de l'itération précédente. Une mémoire à court terme est utilisée pour éviter tout cycle visitant périodiquement le même optimal local. A partir des derniers mouvements mémorisés, un ensemble de solutions est considéré comme *tabou*. Un mouvement n'est donc réalisable que vers l'une des solutions voisines qui ne sont pas taboues.
Le choix des solutions taboues peut être tout simplement les dernières solutions

visitées, mais en général le critère déterminant les solutions taboues à un moment précis de la recherche dépend du problème à optimiser et de la représentation d'une solution. Supposons qu'une solution soit représentée par un ensemble de N couples (V_i, v_i), V_i et v_i étant respectivement la variable numéro i et une valeur possible pour cette variable. Supposons ensuite, qu'un mouvement élémentaire correspond à augmenter (ou à diminuer) la valeur d'une des variables V_i en lui affectant la valeur v_i' la plus proche de v_i dans le domaine de V_i. Il y a alors 2^N solutions proches ($i.e\ max_{1 \leq i \leq N} |V_i' - V_i| \leq 1$) d'une solution visitée. Si le nombre de variables est suffisamment important, le nombre d'itérations où la RT peut rester proche d'un même optimum local, sans passer par une solution déjà visitée, peut devenir supérieur au nombre maximum d'itérations autorisé pour la recherche. Pour éviter ce problème, nous avons choisi qu'une solution est taboue si une de ses variables a une valeur identique à une des variables modifiées parmi les t précédentes solutions. t représente dans ce cas la durée de l'état tabou en nombre d'itération et la mémoire à court terme enregistre les modifications des variables plutôt que les solutions visitées.
Lorsque des caractéristiques de modification définissent l'appartenance d'une solution à l'ensemble tabou, l'ensemble des solutions interdites à chaque itération peut contenir des solutions meilleures (ou conduisant à des solutions meilleures) que toutes celles déjà visitées. Un mécanisme particulier, appelé l'aspiration, permet d'éviter cet inconvénient. Il permet de lever le statut tabou d'une solution, sans pour autant produire de cycle dans le processus de recherche. La fonction la plus simple d'aspiration (et la plus communément utilisée) consiste à annuler le statut tabou d'un mouvement si ce dernier permet d'atteindre une solution de qualité supérieure à celle de la meilleure jusqu'alors.
Cette méthode possède donc un ensemble de mécanismes simples dans leurs principes avec une grande liberté de spécification de ces mécanismes pour l'adapter à un problème donné.

4 Quantification Vectorielle

Par définition, la Quantification Vectorielle (QV) est une technique de classification très utilisée dans le domaine de la compression (Gersho & Gray, 1991). Elle associe à un vecteur d'entrée $\mathbf{x}$ de dimension n un vecteur $\mathbf{y}$ de même dimension appartenant à un dictionnaire $(w_j) j \in [1, \cdots, m']$ qui est un ensemble fini de vecteurs-codes, appelés aussi classes, prototypes ou barycentres puisque ceux-ci sont calculés par une moyenne itérative sur l'ensemble des vecteurs $\mathbf{x}$.
La construction d'un dictionnaire de taille m' est réalisée à partir d'un ensemble d'apprentissage S_a (m vecteurs $\mathbf{x}$ avec $m > m'$). L'algorithme de construction du dictionnaire produit m' prototypes qui minimisent la distorsion par rapport à la base d'apprentissage.

$$distorsion = \frac{1}{m} \sum_{i=1}^{m} \min_{1 \leq j \leq m'} d(\mathbf{x}_i, \mathbf{y}_j) \tag{2}$$

L'algorithme LBG (Gersho & Gray, 1991) est un des algorithmes qui permet la construction du dictionnaire. Il commence avec un prototype «parent» qui représente «grossièrement» S_a (le centroïde de S_a dans notre version de LBG). A chaque itération, il génère à partir d'un prototype «parent» deux nouveaux prototypes «enfant» en modifiant légérement les valeurs du vecteur «parent» dont ils sont issus. Les prototypes «enfant» sont ensuite utilisés pour réaliser un *clustering* de l'ensemble S_a . Chaque *cluster* est alors représenté par un prototype «parent» (le centroïde du *cluster* dans notre version). On obtient donc à partir de k itérations 2^k prototypes.

5 Sélection d'attributs

La sélection d'attributs a pour but principal de ne garder qu'un sous-ensemble d'attributs qui caractérisent les différents objets d'un problème de reconnaissance. Dans le domaine de la classification, cette sélection aura pour objectifs : (1) de réduire le coût d'extraction des attributs caractérisant les objets à reconnaitre, (2) d'améliorer le taux de reconnaissance d'un classificateur, (3) de réduire la complexité de la fonction de décision produite. Pour résoudre le problème de sélection d'attributs, deux grandes familles sont identifiées dans la littérature : L'approche *filter* et l'approche *wrapper* selon qu'elles utilisent ou non le taux de reconnaissance du classificateur (Kohavi & John, 1997).
L'approche *filter* peut être considérée comme une phase de pré-traitement à l'étape d'apprentissage du classificateur. Elle se contente d'estimer le pouvoir discriminant d'un espace d'attributs à partir de mesures statistiques sur l'échantillon d'apprentissage (covariance, corrélation, apport d'information,...). La sélection optimale devra donc maximiser ce pouvoir discriminant. Dans le cas où l'estimateur du pouvoir discriminant possède des propriétes de monotonie, des algorithmes basés sur la technique du *branch and bound* (Kudo & Sklansky, 2000) permettent une sélection rapide d'un sous-ensemble optimal d'attributs.
L'approche *wrapper* considère que le taux de reconnaissance d'un classificateur caractérise le pouvoir discriminant d'un sous-ensemble d'attributs (Kohavi & John, 1997). Un sous-ensemble d'attributs n'est donc optimal qu'en fonction du choix d'un classificateur. De plus, le taux de reconnaissance n'a généralement aucune propriéte de monotonie.
Si la première approche à l'avantage d'être beaucoup plus rapide, car il n'y a qu'un apprentissage pour un ensemble d'attributs préalablement choisi, elle ne permet généralement pas d'obtenir les meilleurs taux de classification par rapport à la seconde. Par contre, la seconde approche se heurte rapidement à un problème d'explosion combinatoire dans le cas d'une évaluation systématique de tous les sous-ensemble productibles, dès que le nombre d'attributs est élevé. Un grand nombre d'algorithmes sous-optimaux existent afin de réaliser cette sélection avec des temps exploitables : Sequential Floating Forward Selection (SFFS), algorithmes génétiques, recherche tabou ... (Kudo & Sklansky, 2000) (Korycinski *et al.*, 2004).

6 Estimation de la généralisation des classificateurs

Un grand nombre d'algorithmes de classification ont été proposés dans les dernières décennies : kppv, réseaux de neurones, arbres de décision, SVM, ... Généralement, ils ont un ensemble de paramètres internes (par exemple C et ceux de la fonction noyau pour les SVM) et externes (sélection d'attributs,...) à déterminer pour un problème donné. Le même algorithme d'apprentissage produira donc un ensemble de fonctions de décision h_θ en fonction des valeurs θ pour ces paramètres. Il est alors indispensable de pouvoir estimer les performances en généralisation des différentes fonctions de décision pour choisir la plus appropriée. On désigne généralement la recherche des valeurs optimales de θ par le terme sélection du modèle. En théorie, l'erreur de généralisation Err_g peut être calculée de la façon suivante :

$$\mathrm{Err}_g = \int L\left(h_\theta(\mathbf{x}), y\right) p(\mathbf{x}, y) d\mathbf{x} dy \tag{3}$$

$L(y_h, y_t)$ est une fonction qui représente le coût d'avoir prédit la classe y_h au lieu de la classe y_t et $p(\mathbf{x}, y)$ correspond à la probabilité que l'objet caractérisé par $\mathbf{x}$ soit de la classe y. En pratique, la valeur Err_g ne peut être calculée, car les données du problème sont représentées par un échantillon $S = \{(\mathbf{x_1}, y_1), \cdots, (\mathbf{x}_m, y_m)\}$ de m exemples et qu'il est impossible d'en déduire directement une formulation exacte de $p(\mathbf{x}, y)$. Par contre, l'erreur empirique pour une fonction de décision h_θ peut être facilement déterminée à partir de S par le calcul de

$$\mathrm{Err}^m_{emp}\left(h^S_\theta\right) = \frac{1}{m} \sum_{i=1}^{m} L\left(h^S_\theta(\mathbf{x}_i), y_i\right) \tag{4}$$

Malheureusement ce n'est pas un bon estimateur de Err_g. En effet, si l'apprentissage est réalisé directement à partir de l'ensemble S, il est facile de construire un classificateur produisant une fonction de décision avec une erreur empirique sur S nulle, mais dont la valeur sur un autre echantillon S' représentatif du même problème soit beaucoup plus élevée. Ceci sera d'autant plus vrai que l'espace des hypothèses induites par l'ensemble des fonctions de décision productibles par le classificateur en fonction des différentes valeurs de θ sera vaste et flexible (Vapnik, 1998).

Pour éviter ce phénomène de sur-apprentissage, différentes techniques ont été proposées (Joachims, 2000). Globalement, elles réalisent toutes leur apprentissage sur des sous-ensembles de S et elles évaluent le risque empirique sur d'autres sous-ensembles de S non utilisés par chacune des phases d'apprentissage. La technique la plus populaire est la validation croisée. Elle consiste à diviser l'échantillon en q parties S_i ($i \in [1..q]$) de tailles identiques, puis à réaliser q apprentissages sur les q échantillons $S \setminus S_i$. L'estimation de Err_g est obtenue à partir de la moyenne des erreurs empiriques calculées sur chaque sous-ensemble S_i non utilisé pendant chaque phase d'apprentissage. L'estimation de l'erreur de «Leave

One Out» (Err_{loo}) est obtenue en choisissant $q = m$ et il a été démontré que Err_{loo} est l'estimateur le moins biaisé (Chapelle *et al.*, 2002) de la valeur de Err_g. La technique de «Leave one out»(*loo*) n'est réellement utilisée que lorsque le nombre d'exemples est très réduit, car son coût en temps de calcul devient rapidement prohibitif. Plus le nombre d'exemples augmente et plus q est choisi petit par rapport à m. Lorsque le nombre d'exemples devient important, il est courant de diviser S en deux ensemble S_a et S_v, respectivement de taille 2/3 et 1/3 de celle de S. S_a est appelé la base d'apprentissage, S_v la base de validation et il n'y a qu'un apprentissage. Cette méthode peut s'identifier à une procédure de validation croisée avec $q = 3$, sous l'hypothèse que la variance sur l'estimation de l'erreur de généralisation est faible. Ce qui justifie de n'avoir qu'un apprentissage au lieu de 3.
Les SVM ont un avantage par rapport à d'autres classificateurs pour l'estimation de Err_g. En effet, une borne supérieure de Err_{loo} peut facilement être déterminée à partir d'un seul apprentissage en fonction du nombre de vecteurs de support et de leurs positions par rapport à la marge. La borne la plus simple à établir est $\mathrm{Err}_{loo} < n_{\mathrm{VS}}/m$ avec n_{VS} le nombre de vecteurs de support. En effet, la minimisation de $\mathcal{W}(\alpha)$ pour S et $S \setminus (\mathbf{x}_i, y_i)$ produira la même fonction de décision si l'exemple i n'est pas un vecteur de support. Par conséquent, chaque exemple qui n'est pas vecteur de support sera correctement classé dans le calcul de Err_{loo}. La borne précédente est directement obtenue avec l'hypothèse que tout vecteur de support peut être mal classé dans le calcul de Err_{loo}. D'autres bornes de Err_{loo} plus précises peuvent être déterminées en tenant compte de la position des vecteurs de support par rapport à la marge (Joachims, 2000).
Un autre point important pour l'estimation de Err_g réside dans le choix de la fonction de coût. La plus simple généralement utilisée est $L(y_h, y_t) = 0$ si y_h est égal à y_t et 1 sinon. Si elle semble tout à fait justifiée sans connaissance particulière sur le domaine (par exemple, il est préférable de faire une fausse détection de panne dans un avion que le contraire), elle peut produire des résultats aberrants lorsque le nombre d'exemples pour chaque classe est très différent. Prenons l'exemple d'une base comportant 99,9% d'exemples de la classe 1 et 0,1% de la classe 2. Une fonction de décision prédisant la classe 1 quelque soit la valeur de $\mathbf{x}$ a une Err_{loo} de 0,1% et une fonction prédisant à 90% la bonne classe d'un exemple quelque soit sa classe a une Err_{loo} de 10% ! Pour éviter ce problème, la fonction de coût choisie n'est pas symétrique et pénalise plus les erreurs de classification sur les exemples de classes minoritaires : $L(y_h, y_t) = m/(m_t c)$ quand $y_h \neq y_t$ avec m_t le nombres d'exemples de classe t dans S et c le nombre total de classes.

7 Nouvelle méthode d'apprentissage pour les SVM

Le succès des SVM a entraîné le développement de nombreux algorithmes permettant leur mise en œuvre. Parmi ceux-ci, l'algorithme SMO (Sequential Minimal Optimization) est souvent considéré comme le plus efficace, bien que la recherche soit toujours très active pour proposer des algorithmes encore plus

performants (Loosli *et al.*, 2004). Dans son article (Platt, 1999), Platt teste l'algorithme SMO sur plusieurs bases d'apprentissage pour évaluer la complexité de son algorithme. Ces résultats montrent que la complexité en temps de calcul se situe entre $O(n^{1,6})$ et $O(n^{2,1})$. Comme la complexité de SMO est aussi liée au nombre de vecteurs de support et que l'erreur de généralisation est reliée au rapport n_{VS}/m (cf. section 6), on en déduit que la complexité est dépendante de la base utilisée et qu'elle est d'autant plus élevée que le problème est difficilement séparable. D'autre part, bien que les SVM soient performants en généralisation, ils ont pour défaut d'avoir une fonction de décision dont la complexité dépend aussi du nombre de vecteurs de support. Par contre, le gain sur la performance en généralisation diminue rapidement avec l'augmentation de cette complexité. De plus, lorsque les valeurs θ du modèle sont éloignées de l'optimal, on observe en général une augmentation du nombre de vecteurs de support. Ce qui se traduit par une augmentation des temps de calcul nécessaires pour produire au final une fonction de décision de faible qualité !

Partant de ce constat, nous proposons une nouvelle méthode d'apprentissage qui a pour objectif de réduire les temps nécessaires à la sélection d'un modèle performant , tout en privilégiant la production avec des SVM de fonctions de décision de complexités réduites. En faisant l'hypothèse que l'estimation de l'erreur de généralisation sur une base de taille réduite est aussi un bon estimateur de la performance du modèle sur la base globale qu'elle représente (Lebrun *et al.*, 2004), il devient alors évident que les temps d'apprentissage nécessaires pour accepter ou rejeter un modèle seront considérablement réduits en utilisant une représentation simplifiée. De plus, si la taille de la base n'est pas trop réduite, la fonction de décision produite aura elle-même une complexité réduite, tout en conservant de bonnes performances en généralisation. La difficulté est alors de savoir comment réaliser cette simplification et choisir son importance.

L'idée principale de notre méthode est de remplacer un ensemble d'exemples proches par un seul prototype. L'algorithme LBG produit 2^k prototypes par classe qui miminisent la distorsion (cf. section 4). Minimiser la distorsion correspond simplement à choisir des prototypes qui soient le plus proche possible des exemples qu'ils représentent. Dans le cas où une classe n'a pas assez de représentants pour une valeur de k donnée, l'ensemble des exemples de cette classe sont choisis comme prototypes. La réunion des prototypes des deux classes forme une nouvelle base d'apprentissage S_a^k qui correspond à une version plus ou moins simplifiée de la base initiale S_a. Le choix de la bonne valeur de k à utiliser est dépendant de la taille de la base, de la nature de la base et du compromis entre complexité de la fonction de décision et taux de reconnaissance. k ne pouvant pas être fixé facilement de manière arbitraire, une notion importante dans notre méthode est de considérer k comme un paramètre du modèle θ. La recherche d'un modèle θ optimal consiste donc à déterminer : la valeur de la constante de régularisation C, la valeur de la largeur de bande σ du noyau gaussien, la valeur de k représentant l'importance de la simplification de la base et le sous-ensemble minimum d'attributs à utiliser. La sélection ou non d'un attribut peut être représentée par une variable entière prenant respectivement les valeurs 1

ou 0. Pour que toutes les valeurs des variables du modèle soient entières, nous avons choisi de discrétiser les valeurs possibles pour les hyperparamètres C et σ. Cette technique couramment utilisée avec les SVM est désignée par le terme *grid search* (Chang & Lin, 2001). Il est évident qu'une évaluation systématique pour obtenir le modèle optimal n'est pas envisageable et qu'il est nécessaire d'utiliser une technique de recherche sous-optimale. La méthode tabou nous a paru un bon choix pour sa facilité d'adaptation et sa robustesse. Elle a d'ailleurs déjà été utilisée pour la sélection d'attributs (Korycinski *et al.*, 2004) ou pour le choix des valeurs des hyperparamètres C et σ des SVM (Cawley, 2001) avec de bons résultats dans les deux cas. L'originalité est ici, d'utiliser la recherche tabou pour optimiser l'ensemble des paramètres. En particulier, le choix du niveau de simplification à un moment donné est contrôlé par la méthode tabou. L'objectif est de permettre un rejet rapide des mauvais paramètres à partir d'une représentation simplifiée et de n'utiliser des représentations moins simplifiées que si elles diminuent suffisamment le taux d'erreur. Pour obtenir un tel résultat, la recherche tabou doit commencer avec une représentation fortement simplifiée de la base (k petit), mais il faut aussi que la fonction objectif à optimiser favorise les solutions de complexités réduites. Nous proposons de pénaliser l'estimation du taux d'erreur en fonction de la complexité de la fonction de décision produite. Nous avons choisi que le doublement du nombre de vecteurs de support ou d'attributs utilisés doit correspondre au moins à une amélioration Δ du taux de reconnaissance, la valeur de Δ étant configurable par l'utilisateur.
L'algorithme tabou cherche alors à minimiser la fonction suivante :

$$f(h_\theta^{S_a}) = \mathrm{Err}\left(h_\theta^{S_a}\right) + c_{p1}\log_2\left(n_{\mathrm{VS}}(h_\theta^{S_a})+1\right) + c_{p2}\log_2\left(n_{au}(h_\theta^{S_a})+1\right) \quad (5)$$

avec S_a la base d'apprentissage initiale, $h_\theta^{S_a}$ la fonction de décision produite par un SVM entrainé à partir du modèle θ, $\mathrm{Err}\left(h_\theta^{S_a}\right)$ l'erreur estimée en généralisation, $n_{\mathrm{VS}}(h_\theta^{S_a})$ le nombre de vecteurs de support utilisés par $h_\theta^{S_a}$ et $n_{au}(h_\theta^{S_a})$ le nombre d'attributs utilisés par $h_\theta^{S_a}$. c_{p1} et c_{p2} permettent de régler l'importance de la pénalité (*i.e.* la valeur de Δ) à appliquer en fonction de la complexité de la fonction de décision. Pour l'estimation de l'erreur de généralisation, il est important de tenir compte des variations possibles du nombre de représentants d'une classe par rapport à l'autre classe en fonction du niveau de simplification. En effet, si initialement les classes 1 et 2 ont repectivement n_{c_1} et n_{c_2} représentants dans la base S_a, dans les versions simplifiées S_a^k le nombre de représentants par classe est repectivement de $n_{c_1}^k$ et $n_{c_2}^k$ avec $n_{c_i}^k = min(2^k, n_{c_i})$. Le rapport $n_{c_1}^k/n_{c_2}^k$ n'est pas constant si n_{c_1} est différent de n_{c_2} (cf. section 8, cas de la transformation d'un problème multiclasse en un ensemble de problèmes binaires) et les variations lorsque k augmente sont d'autant plus importantes que la différence est grande. La fonction de coût $L(.,.)$ doit donc tenir compte du nombre de représentants de chaque classe pour un niveau de simplification donné. A partir des remarques faites à la section 6, nous avons choisi pour le calcul de l'erreur empirique d'utiliser la fonction de coût suivante : $L(y_h, y_t) = (n_{c_h}^k + n_{c_t}^k)/(2n_{c_t}^k)$. Le calcul de la borne supérieure de Err_{loo} doit aussi tenir compte du nombre

de représentants de chaque classe : $\text{Err}_{loo} \leq (n_{\text{VS}_{c_1}}/n^k_{c_1} + n_{\text{VS}_{c_2}}/n^k_{c_2})/2$ avec $n_{\text{VS}_{c_1}}$ et $n_{\text{VS}_{c_2}}$ repectivement le nombre de vecteurs de support appartenant aux classes 1 et 2.
La fonction noyau utilisée est une fonction gaussienne exploitant la sélection d'attributs :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\sum_{l=1}^{n} \beta^l \left(x_i^l - x_j^l\right)^2}{\sigma^2}\right) \tag{6}$$

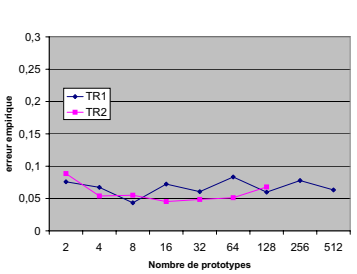
avec n le nombre d'attributs, x_i^l la valeur de l'attribut l de l'exemple i et β^l égal à 0 ou 1 suivant que l'attribut l est ou n'est pas sélectionné dans le modèle θ.
Les bornes minimum et maximum pour les différentes variables du modèle $\theta = (k, log_2(C), log_2(\sigma), \beta^1, \ldots, \beta^n)$ ont été fixées à : $0 \leq k \leq k_{max}$, $-5 \leq log_2(C) \leq 15$, $-10 \leq log_2(\sigma) \leq 10$, $0 \leq \beta^l \leq 1$ $(l \in [0, \ldots, n])$ avec k_{max} la valeur de k correspondant à une non simplification de la base d'apprentissage. Les choix des bornes de $log_2(C)$ et $log_2(\sigma)$ ont pour origine la technique de *grid search* (Chang & Lin, 2001)) . Pour la méthode tabou, ces variables ne sont pas différenciées et le modèle $\theta = (\theta^1, \ldots, \theta^{n'})$ contient n' $(n' = n + 3)$ variables entières. A partir de cette dernière représentation de θ, un mouvement réalisable pour la RT correspond à augmenter ou à diminuer de 1 une des variables θ^i $(i \in [0, \ldots, n'])$ tout en respectant les contraintes de maximalité et de minimalité. Reste à définir, pour la RT, le nombre maximum d'itérations iter_{max} et la définition des solutions taboues. Le nombre iter_{max} doit dépendre de la taille de l'espace de recherche, mais ne pas entrainer une explosion combinatoire. Il a été défini de la façon suivante : $\text{iter}_{max} = 4\sum(\max(\theta^i) - \min(\theta^i) + 1)$ avec $i \in [1, ..., n']$. Nous avons choisi qu'une solution soit taboue si elle contient au moins une variable ayant une valeur taboue. Une valeur pour une variable devient taboue suite à un mouvement l'ayant modifié (cf. section 3). L'ancienne valeur de la variable avant le mouvement est tabou pour cette variable pendant $\text{iter}_{max}/16$ itérations. La RT utilise aussi l'aspiration (cf section 3).

8 Expérimentations

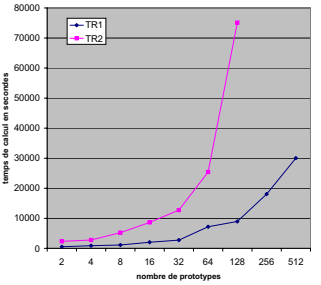
bases	**exemples (m)**	**classes (c)**	**attributs (n)**
ClassPixels	224636	3	33
OpticDigits	4123	10	64
Shuttle	58000	6	9

TAB. 1 – Description des bases.

Trois bases d'apprentissage ont été utilisées (Tab. 1). Deux proviennent de *UCI repository* (Blake & Merz, 1998) : *Shuttle* et *OpticDigits*, elles sont régulièrement



(a) Erreur empirique



(b) Temps de calcul en secondes

FIG. 1 – La «resimplification» (TR2) ou non (TR1) à chaque changement des attributs utilisés. Mesures réalisés avec la base *ClassPixels*

	c_p= **0,01**					*cp*=**0,00**				
FD	T_{app}	k	n_{VS}	n_{att}	Err_{emp}	T_{app}	k	n_{VS}	n_{att}	Err_{emp}
1	207	4	7	2	0,477%	38106	15	127	3	0,013%
2	67	0	2	1	0,007%	14062	10	20	3	0,000%
3	45	0	2	1	0,038%	7948	11	38	3	0,000%
4	152	5	9	2	0,125%	31027	14	63	4	0,035%
5	44	3	2	1	0,018%	36637	7	13	2	0,000%
6	113	2	5	1	0,021%	394	6	24	6	0,000%

TAB. 2 – Résultats avec la base *Shuttle*

utilisées comme référence pour la classification. La troisième : *ClassPixels* provient d'un problème de classification de pixels (Meurie *et al.*, 2003). Les SVM sont par nature des classificateurs binaires. Comme nous ne nous intéressons pas ici à la manière de les combiner pour produire un classificateur multiclasse, nous transformons chaque base en c problèmes de classification binaire, chaque problème correspondant à la discrimination d'une classe i ($i \in [1, \ldots, c]$) par rapport aux $c-1$ autres classes[1] (dans les tableaux de résultats, FD n° i désigne la fonction de décision qui réalise la discrimination de la classe i par rapport aux autres).

Pour éviter tout biais lors de l'évaluation de la qualité d'une fonction de décision, la sélection du modèle est réalisée à partir d'une base S ne contenant que les 2/3 de la base initiale. Le tiers restant forme la base de test St servant à l'évaluation de cette qualité. Dans le cas de l'estimation de l'erreur empirique Err_{emp} la base S est elle-même séparée aléatoirement dans des proportions de 2/3 et 1/3 pour produire respectivement la base d'apprentissage S_a et la base de validation S_v. Ceci afin d'éviter tout phénomène de sur-apprentissage (cf. section 6). Lors de

[1]Cette décomposition est réalisée dans le schéma multiclasse *one versus all* (Hsu & Lin, 2002)

	c_p= **0,01**					*cp*=**0,00**				
FD	T_{app}	k	n_{VS}	n_{att}	Err_{emp}	T_{app}	k	n_{VS}	n_{att}	Err_{emp}
1	3971	3	4	5	3,97%	10666	5	19	14	3,98%
2	9497	4	20	7	12,49%	10905	4	21	18	13,40%
3	5089	3	9	6	8,24%	9715	4	19	18	8,11%

TAB. 3 – Résultats avec la base *ClassPixels*

	cp= **0,01**					*cp*=**0,00**				
FD	T_{app}	k	n_{VS}	n_{att}	Err_{emp}	T_{app}	k	n_{VS}	n_{att}	Err_{emp}
1	212	4	5	16	0,073%	395	4	16	64	0,000%
2	694	4	23	29	0,293%	2406	6	95	21	0,000%
3	288	4	10	18	0,073%	328	4	29	31	0,000%
4	221	2	7	14	1,299%	335	4	19	25	0,723%
5	271	4	10	19	0,000%	531	4	27	62	0,649%
6	242	4	8	14	0,073%	416	5	29	17	0,000%
7	248	2	7	22	0,000%	4169	4	32	64	0,000%
8	254	3	9	23	0,220%	330	4	12	27	0,220%
9	767	6	32	22	2,146%	4597	8	193	59	2,146%
10	563	4	32	29	1,755%	575	4	32	29	1,682%

TAB. 4 – Résultats avec la base *OpticDigits*

la RT pour la sélection d'un bon modèle θ les attributs utilisés peuvent changer. Normalement à chaque modification d'un β^l la simplification de S_a par la QV devrait être recalculée. Pour autant, cette simplification ne sera appliquée, pour une valeur de k donnée, qu'une fois et ceci avec l'ensemble des attributs. Les prototypes produits ne sont donc pas optimaux au sens de l'algorithme LBG, mais suffisamment si on prend en considération le coût supplémentaire en temps de calcul de la «resimplification» (cf. Fig. 1(a),1(b)). Nous insistons aussi sur le fait que dans notre méthode la représentation à un niveau k ne sera calculée que si la recherche tabou l'utilise, le gain en temps de calcul n'étant pas négligable lorsque k peut devenir important.

Dans toutes les expériences, nous avons choisi les mêmes valeurs pour les coefficients c_{p1} et c_{p2} désignés génériquement par le terme c_p. Les tableaux 2, 3, 4 donnent pour chaque problème binaire le temps d'apprentissage T_{app} en secondes pour la selection d'un modèle θ, le niveau k de simplification, le nombre de vecteurs supports n_{VS}, le nombre d'attributs sélectionnés n_{att} et l'erreur empirique Err_{emp} pour différentes valeurs de c_p.

A la lecture du tableau 2, on observe que l'apprentissage des 6 fonctions de décision dure seulement 10 minutes avec notre méthode ($c_p = 0,01$) alors que la simple recherche d'un modèle optimal qui ne contient que C et σ avec une technique de *grid search* dure plus de 400 heures (Lebrun *et al.*, 2004) avec

	c_p= **0,01**					*cp*=**0,00**				
FD	T_{app}	k	n_{VS}	n_{att}	$\widehat{Err}_{loo}$	T_{app}	k	n_{VS}	n_{att}	$\widehat{Err}_{loo}$
1	669	8	10	17	0,877%	1341	9	15	22	0,804%
2	1659	8	24	27	1,026%	1549	9	37	28	0,953%
3	1725	8	29	32	2,064%	1638	9	25	34	1,471%
4	3294	9	57	33	2,901%	3420	9	56	42	2,974%
5	1578	9	36	26	0,943%	2162	10	24	30	1,822%
6	1478	9	26	21	1,690%	1799	10	19	29	2,950%
7	1601	9	13	20	2,494%	1288	9	17	26	1,050%
8	2975	10	14	20	2,315%	2775	10	13	25	1,383%
9	8251	10	74	35	5,325%	8670	10	73	35	3,918%
10	1872	8	42	20	8,489%	291	6	6	10	11,561%

TAB. 5 – Résultats avec la base *OpticDigits*

l'intégralité de la base *Shuttle*. On remarque en particulier que l'utilisation du coefficient de pénalité c_p réduit les temps d'apprentissage et la complexité des fonctions de décision sans trop dégrader la performance de ces fonctions. Le tableau 3 illustre que l'on peut produire des fonctions de décision exploitables dans le cadre de la classification de pixels, alors qu'elles ne le sont pas avec une technique d'apprentissage classique (Lebrun *et al.*, 2004).

Nous avons aussi essayé d'utiliser la valeur de $\widehat{Err}_{loo}$ comme estimateur de l'erreur de généralisation pour la sélection d'un modèle θ. Les résultats (*cf.* Tab. 5) obtenus sont moins bons en général qu'avec l'utilisation de la valeur de Err_{emp} (*cf.* Tab. 4). En particulier, les temps d'apprentissage augmentent rapidement. En analysant les modèles sélectionnés, on observe qu'ils ont tous des valeurs importantes pour la constante de régularisation C. Ceci s'explique par le fait que pour avoir un ratio n_{VS}/m faible, il faut peu d'exemples dans la marge et donc une valeur de C importante. Donc l'utilisation de $\widehat{Err}_{loo} = n_{VS}/m$ comme estimateur de l'erreur de généralisation introduit un autre phénomène de biais qui minimise l'importance de la constante de régularisation C, car tout vecteur de support est considéré comme mal classé dans son calcul.

9 Conclusion

Une nouvelle méthode d'apprentissage avec sélection des attributs sur des bases de taille importante et utilisant des SVM est présentée. L'idée principale repose sur la combinaison de l'utilisation de la QV pour réduire la taille de la base d'apprentissage et de la RT pour la sélection d'un modèle de bonne qualité. La pénalisation de l'estimation de l'erreur de généralisation à travers deux coefficients permet à l'utilisateur de définir le compromis entre qualité et rapidité d'évaluation. Les résultats expérimentaux obtenus montrent que la méthode proposée produit des fonctions de décision performantes en généralisation et dont

la complexité est réduite. De plus, le laps de temps nécessaire à la selection d'un modèle est relativement faible.
Les travaux futurs devront comparer différents metaheuristiques à la RT pour la sélection du modèle et différents méthodes de simplification, ceci afin de rechercher comment notre méthode d'apprentissage est perfectible. Ils devront étendre la méthode à la production de schémas multiclasses à partir de la combinaison de classificateurs SVM binaires et mesurer l'influence de la réduction de la complexité des fonctions de décision sur la qualité globale de différents schémas multiclasses. Il sera aussi important de tester d'autres bornes de l'erreur de *loo* en espérant réduire le phénomène de biais observé.

Références

Blake C. & Merz C. (1998). Uci repository of machine learning databases. advances in kernel methods, support vector learning. In *University of California, Irvine, Dept. of Information and Computer Sciences*.

Cawley G. C. (2001). Model selection for support vector machines via adaptive step-size tabu search. In *ICANNGA*.

Chang C.-C. & Lin C.-J. (2001). Libsvm : a library for support vector machines. Sofware Available at http ://www.csie.ntu.edu.tw/˜cjlin/libsvm.

Chapelle O., Vapnik V., Bousquet O. & Mukherjee S. (2002). Choosing multiple parameters for support vector machines. In *Machine Learning*, volume 46, p. 131–159.

Gersho A. & Gray R. M. (1991). *Vector Quantization and Signal Compression*. Kluwer Academic.

Glover F. (1989a). Tabu search : part i. In *on Computing*, volume 1, p. 190–206.

Glover F. (1989b). Tabu search : part ii. In *on Computing*, volume 2, p. 4–32.

Hao J.-K., Galinier P. & Habib M. (1999). Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. In *Revue d'Intelligence Artificielle*, volume 13, p. 283–324.

Hsu C.-W. & Lin C.-J. (2002). A comparison of methods for multiclass support vector machines. In *IEEE Transactions in Neural Networks*, volume 13, p. 415–425.

Joachims T. (2000). Estimating the generalization performance of a SVM efficiently. In *ICML-00*, p. 431–438.

Kohavi R. & John G. H. (1997). Wrappers for feature subset selection. In *Artificial Intelligence*, volume 97, p. 273–324.

Korycinski D., Crawford M. M. & Barnes J. W. (2004). Adaptive feature selection for hyperspectral data analysis. In *SPIE*, volume 5238, p. 213–225.

Kudo M. & Sklansky J. (2000). Comparison of algorithms that select features for pattern classifiers. In *Pattern Recognition*, volume 33, p. 25–41.

Lebrun G., Charrier C. & Cardot H. (2004). SVM training time reduction using vector quantization. In *ICPR*, volume 1, p. 160–163.

Loosli G., Canu S., Vishwanathan S. V. N., Smola A. J. & Chattopadhyay M. (2004). Une boîte à outils rapide et simple pour les svm. In *CAP 04*, p. 113–128.

Meurie C., Lebrun G., Lezoray O. & Elmoataz A. (2003). A comparison of supervised pixels-based color image segmentation methods. application in cancerology. In *WSEAS Transactions on Computers*, volume 2, p. 739–744.

Platt J. (1999). *Fast Training of Support Vector Machines using Sequential Minimal Optimization, Advances in Kernel Methods-Support Vector Learning*. MIT Press, pp. 185-208.

Vapnik V. N. (1998). *Statistical Learning Theory*. New York, wiley edition.

Exploration de l'espace de paramètres d'un modèle à base d'agents

Benoît Calvez, Guillaume Hutzler

Université d'Evry-Val d'Essonne/CNRS
LaMI, UMR 8042
523, Place des Terrasses 91000 Evry, France
bcalvez@lami.univ-evry.fr et hutzler@lami.univ-evry.fr

Résumé :

Quand on développe un système multi-agent (SMA) ou des modèles dans le contexte de la simulation à base d'agents, la mise au point du modèle constitue un pas important dans le processus de conception. En effet les modèles à base d'agents sont généralement caractérisés par de nombreux paramètres qui déterminent la dynamique globale du système. De plus, des changements même faibles sur certains paramètres (parfois un seul) peuvent mener quelquefois à une modification radicale de la dynamique du système dans son ensemble. Le développement d'un modèle à base d'agents et le choix de ses paramètres peuvent alors devenir longs et fastidieux en l'absence de stratégie précise, automatique, systématique pour explorer cet espace de paramètres.

C'est au développement d'une telle stratégie que nous travaillons en suggérant l'utilisation des algorithmes génétiques. L'idée est de capturer dans la fonction de fitness le but du processus de conception (efficacité pour le SMA de réaliser une fonction donnée, réalisme pour les modèles à base d'agents…) et de faire évoluer automatiquement le modèle dans cette direction. Cependant l'utilisation des algorithmes génétiques dans le contexte de la simulation à base d'agents apporte des difficultés spécifiques que nous allons développer dans cet article, en expliquant des solutions possibles et en les illustrant sur un modèle simple et bien connu : le fourragement par une colonie de fourmis.

Mots-clés : Systèmes multi-agents et systèmes distribués, Modélisation & Simulation, Algorithmes génétiques

1 Introduction

La simulation à base d'agents (SBA) s'intéresse à la modélisation et la simulation de systèmes complexes. Son but est de reproduire la dynamique d'un système réel en modélisant les entités par des agents, dont le comportement et les interactions ont été définis. Une première validation d'un tel modèle est obtenue en comparant la dynamique résultante du modèle simulé avec celle du système réel (mesurée grâce aux données

expérimentales). De manière similaire, les systèmes multi-agents (SMA) sont conçus pour accomplir une fonction donnée de façon collective et décentralisée. Le système ne peut alors être validé que si la fonction est réalisée et si elle est efficace.

Dans les deux cas, un des aspects importants dans le processus de conception est lié à la mise au point des paramètres du modèle. En effet, la nature du modèle est généralement caractérisée par de nombreux paramètres qui déterminent la dynamique globale du système. L'espace de paramètres est alors gigantesque. De plus, le comportement de ces systèmes complexes est souvent chaotique : d'un côté de petits changements faits sur un seul paramètre peuvent mener quelquefois à une modification radicale de la dynamique du système entier : d'un autre côté, quelques phénomènes émergents se produisent seulement dans des conditions très spécifiques et ne se produiront pas si ces conditions ne sont pas rencontrées. L'espace de solution peut alors être très petit. Comme conséquence, le développement et le réglage des paramètres d'un modèle à base d'agents peuvent devenir longs et fastidieux si nous n'avons pas de stratégie précise, automatique et systématique pour explorer l'espace de paramètres.

L'approche que nous suggérons est de considérer le problème de développement et de validation des modèles SBA ou SMA comme un problème d'optimisation. La validation peut alors être reformulée comme l'identification d'un jeu de paramètres qui optimise une fonction. La fonction d'optimisation pour la SBA pourrait être la distance entre le modèle artificiel que nous simulons et le système réel. La fonction d'optimisation pour un SMA pourrait être l'efficacité dans la réalisation d'une fonction. Considérant la grande dimension du problème, des techniques d'optimisation telle que les algorithmes génétiques peuvent être alors utilisées pour explorer l'espace de paramètres et trouver le meilleur jeu de paramètres en respectant la fonction d'optimisation.

Cependant l'utilisation des algorithmes génétiques dans ce contexte n'est pas simple. La première raison tient dans le choix de la fonction de fitness (fonction d'évaluation) : les systèmes ou les simulations à base d'agents sont dynamiques et souvent caractérisés par des phénomènes émergents et transitoires ce qui complique la mesure de la fonction de fitness. La ou les caractéristiques du système devant être mesurées, ainsi que l'instant de la mesure sont des facteurs qui influencent fortement les résultats de l'algorithme génétique. Si la fonction de fitness n'est pas choisie avec soin, les modèles résultants seront optimisés pour cette fonction spécifique, ce qui peut ne pas correspondre aux buts initiaux du concepteur du modèle. La seconde raison est qu'aucun modèle mathématique ne permet d'anticiper la dynamique d'un modèle à base d'agents sans l'exécuter. Le calcul de la fonction de fitness requiert alors l'exécution du système ou de la simulation multi-agent (et même plusieurs exécutions pour prendre en compte la stochasticité du simulateur), ce qui implique un fort coût de calcul. Il est alors nécessaire de développer des stratégies pour accélérer la convergence de l'algorithme.

En section 2, nous présentons la problématique du réglage des paramètres d'une simulation à base d'agents. Puis, en section 3, nous présentons la structure générale des algorithmes génétiques et montrons les difficultés qu'apporte l'application de ces techniques à la simulation multi-agent. En section 4, nous proposons un guide pour l'utilisation des algorithmes génétiques avec la simulation à base d'agents et montrons comment cela peut s'appliquer dans l'exemple du fourragement par une colonie de fourmis avant de conclure en section 5.

2 Réglage des paramètres

2.1 Les paramètres des modèles à base d'agents

Dans le contexte de la simulation à base d'agents, un modèle et son simulateur correspondant incluent de nombreux paramètres. Ces paramètres sont de natures différentes. Quelques paramètres sont propres au simulateur : le pas de discrétisation pour la modélisation du temps et de l'espace par exemple peut être une caractéristique fixée du simulateur. Une des conséquences est que ces paramètres peuvent généralement ne pas être modifiables par l'utilisateur. Pour cette raison, nous n'incluons pas ce type de paramètre dans notre espace de recherche. Nous incluons seulement les paramètres qui sont spécifiques au modèle. Certain d'entre eux peuvent être extraits de la connaissance du domaine (expérimentale ou théorique) et peuvent alors être associés à des valeurs fixes. D'autres paramètres doivent être gardés comme variables. Par exemple, nous souhaitons tester une hypothèse sur un modèle : nous mettons alors le ou les paramètres correspondant à cette hypothèse en variable.

Par ailleurs, la connaissance peut ne pas être directement compatible avec le modèle. Il peut, par exemple, y avoir un problème de niveau : un modèle peut être conçu de façon assez fine, et seulement des connaissances sur le comportement global du modèle peuvent être connues.

Dans ce cas, une approche simple peut être d'essayer quelques valeurs et de simuler le modèle pour voir comment il se comporte globalement. Ce que nous proposons est d'avoir une approche pour automatiser ce long et fastidieux processus.

2.2 Objectif

Selon la motivation du travail du concepteur, les critères utilisés pour explorer l'espace de paramètres seront eux aussi différents. La motivation peut être de modéliser et de simuler un système réel, mais cela peut être aussi d'étudier les modèles discrets qui peuvent produire un phénomène émergent donné. Finalement, la motivation peut être de proposer des modèles qui réalisent une meilleure performance dans la réalisation d'une fonction spécifique.

Dans le premier cas, nous voulons vérifier si le modèle artificiel simule correctement le comportement du système réel. La validation du modèle sera alors d'avoir un comportement identique (aussi près que possible) à la connaissance expérimentale. Le problème de recherche peut être vu comme la recherche du jeu de paramètres qui minimise la distance entre les données réelles et les données simulées.

Avoir un comportement similaire peut aussi signifier que des phénomènes émergents connus pour se passer dans la vie réelle peuvent être observés dans la simulation. Des lignes de fourmis émergentes, par exemple se produisent seulement si le chemin de phéromones a des propriétés spécifiques, comme nous le verrons dans la section suivante. L'émergence de ce phénomène sera alors associée à des valeurs spécifiques de paramètres, et le problème consistera à rechercher un certain nombre d'intervalles de paramètres où un phénomène est observable. Dans quelques cas, choisir des valeurs légèrement différentes peut mener à des résultats complètement différents durant la simulation, ce qui complique l'exploration manuelle de l'espace de paramètres et justifie

le développement de techniques automatiques.

2.3 Exemple

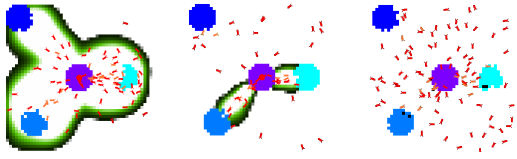
Nous présentons la mise au point des paramètres d'un modèle à base d'agents avec l'exemple du fourragement de fourmis (nous utilisons la plateforme programmable multi-agents `NetLogo` (Tisue & Wilensky, 2004) avec son modèle « Ants »). Le principe est que chaque fourmi cherche de la nourriture et la rapporte au nid en sécrétant sur le chemin du retour une substance chimique appelé phéromone. Quand d'autres fourmis sentent ces dernières, elles suivent le chemin ainsi tracé jusqu'à la source de nourriture, ce qui renforce la présence de phéromones. Nous pouvons voir émerger des lignes de fourmis, qui sont similaires à ce que l'on observe dans des conditions naturelles. Dans le modèle que nous utilisons, il y a un nid au centre de la zone de simulation, et trois sources de nourritures autour du nid.

Dans ce modèle, deux paramètres conditionnent la formation de chemins de phéromones. Le premier est le taux de diffusion des phénomènes qui correspond au fait qu'une proportion donnée de phéromones sera diffusée aux zones voisines au prochain pas de simulation. Ceci est utilisé pour simuler la diffusion des phéromones dans l'atmosphère. Le second paramètre est le taux d'évaporation des phéromones, ce qui correspond au fait qu'une proportion donnée de phéromones disparaîtra de la zone au prochain pas de simulation. Ceci est utilisé pour simuler l'évaporation des phéromones dans l'atmosphère.

Si nous changeons le second paramètre, nous pouvons avoir différents comportements de simulation. La table 1 montre la variation du taux d'évaporation.

	Modèle 1	Modèle 2	Modèle 3
Taux de diffusion	50	50	50
Taux d'évaporation	0	15	99

TAB. 1 – Modèles avec différents taux d'évaporation.



Modèle 1 Modèle 2 Modèle 3

FIG. 1 – Des résultats de simulation pour les trois modèles du tableau 1

La figure 1 montre les résultats durant la simulation. Nous obtenons deux dynamiques globales différentes pour le système. La première dynamique (simulations 1 et 3) est une recherche aléatoire de nourriture (l'une à cause de la surabondance de phéromones ou l'autre à cause de leur absence). La seconde dynamique (simulation 2) est la recherche de nourriture avec des files de fourmis.

Nous pouvons être intéressés plus précisément dans la dynamique des files de fourmis. Le tableau 2 montre trois modèles avec de petites modifications pour les deux

paramètres. Nous pouvons voir dans la figure 2 que ces petites variations peuvent mener à différentes dynamiques : les différences tiennent à la façon dont les sources de nourritures sont exploitées. Dans le modèle 4, des sources de nourritures sont exploitées à tour de rôles alors que dans le modèle 6, elles sont exploitées toutes en même temps. Comme résultat, nous observons une, deux ou trois lignes de fourmis.

	Modèle 4	Modèle 5	Modèle 6
Taux de diffusion	40	50	60
Taux d'évaporation	15	15	20

TAB. 2 – Modèles avec des paramètres légèrement différents.

Modèle 4 Modèle 5 Modèle 6

FIG. 2 – Des résultats de simulation pour les trois modèles du tableau 2.

2.4 Travaux précédents

Différentes méthodes ont déjà été proposées pour explorer automatiquement l'espace de paramètres de modèles discrets. Dans la plateforme `NetLogo` par exemple, l'outil « BehaviorSpace » (Tisue & Wilensky, 2004) permet d'explorer automatiquement et systématiquement l'espace de paramètres. Cet espace est un produit cartésien de valeurs que chaque paramètre peut prendre, certains d'entre eux étant choisis comme fixés, d'autres étant limités à un sous-ensemble de toutes les valeurs possibles. Cependant, quand nous avons beaucoup de paramètres, quelques uns peuvent prendre un grand nombre de valeurs (paramètres de valeurs réels par exemple) et l'espace de paramètres devient alors énorme ce qui interdit une exploration systématique.

D'autres méthodes ont été proposées, qui explorent différentiellement l'espace entier de paramètres, en se concentrant sur les zones les plus intéressantes. C'est le cas de la méthode développée par Brueckner et Parunak (Brueckner & Parunak, 2003). Ces derniers utilisent une infrastructure de balayage des paramètres (« parameter sweep infrastructure »), qui est similaire à l'outil « BehaviorSpace ». Cependant pour éviter une exploration systématique, ils utilisent des agents chercheurs et introduisent la notion de fitness. Le but d'un agent chercheur est de voyager dans l'espace de paramètres à la recherche de la plus haute fitness. Partant d'une zone de l'espace donnée dans l'espace de paramètres, des agents chercheurs ont deux choix : bouger ou simuler. Chaque agent choisit en fonction de la confiance qu'il a sur l'estimation de la fitness (proportionnelle au nombre de simulations en ce point de l'espace) et de la valeur de la fitness. S'il décide de bouger, il se dirige vers les régions voisines qui ont la plus haute fitness. Un désavantage de cette méthode est que des agents chercheurs peuvent se diriger vers un maximum local de fitness.

3 Utilisation des algorithmes génétiques

Comme le réglage des paramètres d'un modèle est un problème fortement combinatoire, nous proposons d'utiliser des algorithmes génétiques qui apportent généralement de bons résultats sur des problèmes de ce genre.

3.1 Principe des algorithmes génétiques

Les algorithmes génétiques sont une famille de modèles informatiques inspirés par l'évolution. Ils permettent de résoudre de nombreuses classes de problèmes, plus spécialement des problèmes d'optimisation. Dans ce cadre, la solution potentielle d'un problème est encodée sur une structure linéaire de données, qui est appelé un chromosome. L'algorithme travaille sur un ensemble de plusieurs chromosomes qui est appelé une population. Des opérateurs sont appliqués à cette population.

La population des chromosomes est initialisée de façon aléatoire. Chaque chromosome est alors évalué en utilisant une fonction de fitness, qui mesure la qualité de cette solution potentielle en fonction du problème initial : cela revient à donner un score pour chaque chromosome.

Une sélection est faite parmi la population de chromosomes : nous obtenons une nouvelle population nommée population parent. Des opérateurs de recombinaison et de mutation sont alors appliqués à cette population : nous obtenons une nouvelle population nommée une population intermédiaire. La recombinaison consiste en l'échange de parties de chromosomes. Avec cette opération, nous obtenons deux nouveaux chromosomes. C'est l'opérateur le plus fréquent dans les algorithmes génétiques. Intuitivement le rôle de cette opération est de prendre la meilleur partie des chromosomes (c'est-à-dire des solutions potentielles) pour obtenir un meilleur chromosome. La mutation consiste à altérer aléatoirement une partie du chromosome. Cette opération évite de converger prématurément vers une solution locale.

Les nouveaux chromosomes de la population intermédiaire sont évalués. Une nouvelle population est finalement créée à partie de la population initiale et la population intermédiaire, avant de recommencer un nouveau cycle dans l'algorithme génétique.

3.2 Choix de la fonction de fitness

Si nous considérons l'exploration de l'espace de paramètres comme un problème d'optimisation, nous devons définir de façon très attentive la fonction qui doit être maximisée par l'algorithme. Cette fonction de fitness est d'une importance fondamentale : les modèles, qui seront sélectionnés, sont ceux qui réalisent la meilleur performance selon cette fonction. Dans le contexte de la simulation à base d'agents, le choix de la fonction de fitness est problématique pour plusieurs raisons : premièrement, ce n'est pas le résultat d'un calcul mais de la dynamique d'un processus qui doit être évalué ; deuxièmement, des phénomènes émergents peuvent être difficiles à caractériser de façon quantitative car ils sont souvent reliés à une interprétation subjective réalisée par un observateur humain.

3.2.1 Quantitative contre qualitative

Valider un modèle à base d'agents par l'évaluation de la distance entre la simulation et le système réel peut être fait soit de façon quantitative, soit de façon qualitative.

Dans le cas quantitatif, les données sont mesurées dans la simulation et comparées aux données mesurées dans des conditions similaires dans le système réel. La distance entre la simulation et le système réel est alors la distance euclidienne entre les deux vecteurs de données. Si nous essayons de sélectionner des modèles qui sont optimisés pour la réalisation d'une certaine fonction, la fonction de fitness peut être alors directement mesurée par la performance du système pour cette fonction. Dans la cas de fourragement par une colonie de fourmis, cela correspondrait à la quantité de nourriture ramenée au nid après une période donnée ou le temps nécessaire pour ramener toute la nourriture au nid.

Dans le cas qualitatif, ce qui est important est qu'un phénomène émergent donné soit présent dans la simulation. La difficulté est alors de traduire ces observations dans une mesure quantitative (la fonction de fitness). Dans l'exemple du fourragement par une colonie de fourmis, nous savons à partir d'observations que les fourmis s'organisent dynamiquement le long de chemins entre le nid et les sources de nourritures, du fait du marquage de ce chemin grâce à des phéromones. La fonction de fitness pourrait alors être conçue pour récompenser des modèles dans lesquels le chemin de phéromones est formé complètement entre le nid et les sources de nourriture. Dans quelques cas, la caractérisation de tels phénomènes émergents peut ne pas être aussi simple puisqu'elle peut être le résultat d'une interprétation subjective réalisée par l'observateur, qui peut ne pas être facilement quantifiable.

3.2.2 Un processus dynamique

Dans un problème d'optimisation classique, la fonction de fitness correspond au résultat d'un calcul. Par conséquent, la question de l'instant de la mesure n'a pas de sens : la mesure est faite quand le calcul est fini. Au contraire, les simulations à base d'agents sont des processus dynamiques qui évoluent au fil du temps et généralement ne finissent jamais.

Nous pouvons clairement voir dans les exemples donnés dans la section précédente que l'évaluation de la fonction fitness doit être généralement faite à un instant donné. Le choix de cet instant n'est pas neutre et peut influencer grandement les performances de l'algorithme génétique et le modèle résultant. Si nous essayons par exemple de sélectionner des modèles qui montrent un comportement donné qui est transitoire, l'évaluation de la fonction de fitness sur un seul pas de simulation peut ne pas être sensible à ce comportement ; il faudra donc répéter la mesure à différents pas de simulation pour que la fonction de fitness puisse prendre en compte ce phénomène.

La figure 3 montre que la simulation de fourragement est un processus dynamique. Par exemple, l'émergence de files de fourmis n'est pas présente pendant toute la simulation. Cet exemple montre la difficulté à choisir l'instant d'évaluation de la fonction de fitness.

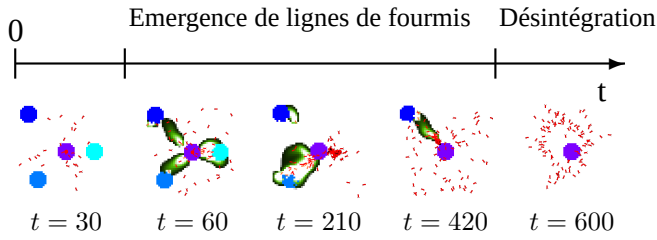


FIG. 3 – Fourragement par une colonie de fourmis à différent pas de simulation

3.3 Calcul de la fonction de fitness

3.3.1 Temps

Comme aucun modèle mathématique ne peut anticiper la dynamique d'un modèle à base d'agents sans l'exécuter, le calcul de la fonction de fitness requiert une, voire plusieurs simulations. Ce qui signifie que le temps requis pour calculer la fonction de fitness serait important (ce qui n'est généralement pas le cas pour des problèmes d'optimisation classique). Le temps global de calcul pour réaliser un algorithme génétique est $(n \times N) \times T_f$ où n est le nombre de chromosomes, N est le nombre de générations et T_f est le temps pour calculer la fonction de fitness. Si T_f est de 10 minutes, n est de 20 chromosomes et N est de 100 générations, alors le temps global pour réaliser l'algorithme génétique est presque de deux semaines. Nous devons par conséquent trouver des méthodes pour réduire soit le nombre de chromosomes, soit le temps pour converger vers un optimum, soit encore le temps pour calculer la fonction de fitness. Nous avons principalement étudié la dernière possibilité à travers le calcul distribué et l'approximation de la fonction de fitness (Jin, 2003).

Comme les différents modèles sont indépendants des uns des autres, l'évaluation de leur fitness est aussi indépendante. Donc chaque évaluation de leur fitness (c'est-à-dire chaque simulation à base d'agents) peut être faite sur différents ordinateurs. Nous pouvons alors avoir plusieurs ordinateurs pour simuler des modèles et utiliser un algorithme génétique parallèle de type maitre-esclave (Cantú-Paz & Goldberg, 2000), qui améliore les performances comparées à un algorithme génétique classique éxécuté sur un seul ordinateur.

L'approximation de la fitness revient à approximer le résultat de la simulation par un modèle mathématique, tel un réseau de neurones ou un polynôme par exemple. Nous essayons cette approche en entraînant un réseau de neurones avec des données de test. Après la phase d'apprentissage, nous utilisons le réseau de neurones pour calculer la fitness, avec une approche de contrôle à base de générations, dans laquelle la population entière de η générations est évaluée avec la fonction de fitness réelle toutes les λ générations (Jin *et al.*, 2002). Cependant les résultats ne sont pas aussi bons et cette approche a été temporairement abandonnée.

3.3.2 Stochasticité

Les résultats de deux simulations à base d'agents fondées sur exactement le même modèle peuvent comporter généralement de légères différences dues à la stochasticité du modèle et du simulateur. Une simulation n'est pas suffisante pour évaluer la fonction de fitness : elle peut seulement être considérée comme une estimation du résultat pour la fitness.

Nous avons étudié la stochasticité du modèle « Ants » et du simulateur `NetLogo`. Nous avons choisi trois fonctions de fitness différentes (expliquées dans les trois exemples de la section 4) et nous avons évalué, en fonction du nombre de simulations, le taux d'erreur dans l'estimation de la fitness comparée à la fitness « réelle » (estimée avec 100 simulations). Le résultat est montré dans le tableau 3.

	Fitness 1	Fitness 2	Fitness 3
Nombre de Simulations	Taux d'erreur	Taux d'erreur	Taux d'erreur
1	$\simeq$34.57%	$\simeq$10.92%	$\simeq$13.28%
5	$\simeq$14.74%	$\simeq$4.85%	$\simeq$6.34%
10	$\simeq$10.3%	$\simeq$3.38%	$\simeq$4.39%
15	$\simeq$8.3%	$\simeq$2.85%	$\simeq$3.58%
20	$\simeq$7.26%	$\simeq$2.39%	$\simeq$3.15%
σ^*	0.41	0.14	0.17

TAB. 3 – Exemple de stochasticité

Nous voyons, par exemple, que nous pouvons obtenir moins de 5% d'erreur sur l'estimation de la fitness 2 en simulant le modèle cinq fois. Mais la stochasticité est plus ou moins importante en fonction de la fitness. La fitness 1, par exemple, est beaucoup plus sensible que la fitness 2.

Dans un tel environnement bruité, une première solution est d'augmenter la taille de la population (Beyer, 2000; Goldberg *et al.*, 1992). Multiplier le nombre de modèles simulés réduit les effets de la stochasticité. Une seconde solution est de simuler chaque modèle plusieurs fois pour améliorer l'évaluation de la fonction de fitness ; ces deux solutions augmentent grandement le nombre de simulations, d'où la durée , de l'algorithme génétique.

Une autre solution est d'utiliser la même technique qu'avec l'approximation de la fonction de fitness. Une solution pour le problème de stochasticité est alors d'estimer la fitness de chaque modèle avec une simulation, et à chaque n générations de l'algorithme génétique (n à choisir selon la stochasticité du modèle et la qualité désirée de l'estimation), d'estimer la fitness de chaque modèle avec x simulations.

Nous utilisons un algorithme génétique élitiste (Beker & Hadany, 2002) c'est-à-dire nous gardons les meilleurs chromosomes durant l'algorithme, ce qui permet d'améliorer continuellement la solution. Notre implémentation de l'algorithme génétique remplace seulement 25% de la population à chaque génération. Toutes les trois générations, nous estimons la fitness des modèles avec plus de simulations.

4 Structure générale et application

Jusqu'à maintenant, nous avons identifié plusieurs difficultés particulières aux systèmes à base d'agents. Nous proposons maintenant une structure générale ou des grandes lignes pour l'application des algorithmes génétiques dans ce contexte très spécifique et montrons avec de nombreux exemples comment cela peut être utilisé :

1. déterminer le but de l'étude ;
2. élaborer un modèle multi-agent ;
3. choisir les paramètres du modèle à faire évoluer par l'algorithme génétique ;
4. choisir la fonction de fitness : qu'est-ce que nous voulons optimiser ? quand et comment devons-nous évaluer la fonction ?
5. étudier la stochasticité du modèle ; simuler le modèle plusieurs fois et étudier les résultats (calculer l'écart-type) ; déterminer la procédure pour l'exploration en conséquence ;
6. étudier le temps de calcul de la simulation ; si la simulation requiert beaucoup de temps, utiliser le calcul distribué ; si la simulation requiert trop de temps, utiliser une approximation de la fitness ;
7. choisir le nombre de chromosomes dans la population ;
8. exécuter l'algorithme génétique.

Nous détaillons maintenant comment ceci peut être appliqué pour l'exemple de fourragement par une colonie de fourmis. Dans les trois exemples, nous utilisons le modèle qui a été présenté dans la section précédente, avec 10 fourmis. La principale différence entre eux est liée à la fonction de fitness.

4.1 Exemple 1

1. notre but est d'optimiser le comportement de fourragement ;
2. le modèle est le modèle « ants » de `NetLogo` ;
3. les paramètres que nous faisons évoluer sont le taux de diffusion et le taux d'évaporation ;
4. la fonction de fitness est la quantité de nourriture ramenée au nid entre le centième et le deux centièmes pas de simulation ;
5. le résultat de l'étude de la stochasticité est montré dans le tableau 3 pour la fitness 1 ; pour évaluer la fonction de fitness, nous utilisons une simulation ; toutes les 3 générations nous utilisons 10 simulations pour évaluer la fonction de fitness de façon plus précise ;
6. l'évaluation de la fonction de fitness (c'est-à-dire une simulation) requiert environ 15 secondes ; nous utilisons seulement un seul ordinateur ; une nuit de calcul est suffisante pour calculer 100 générations ;
7. nous prenons 20 chromosomes ;
8. nous exécutons l'algorithme génétique pour 100 générations.

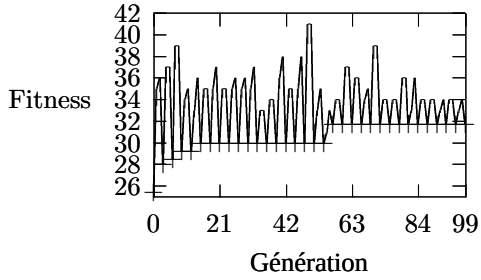


FIG. 4 – Résultat de l'algorithme génétique pour l'exemple 1

La figure 4 montre les résultats. À la fin des 100 premiers pas de simulation, les lignes de fourmis sont construites. Durant les 100 pas de simulations suivants, les fourmis exploitent les sources de nourriture en utilisant les chemins de phéromones. Les meilleurs modèles sont ceux où les trois sources de nourritures sont exploitées en même temps. Cependant le taux d'évaporation est plutôt faible (le taux d'évaporation est égal à $8,1\%$ et le taux de diffusion est égal à $88,6\%$). En conséquence, quand il n'y a plus de nourriture dans une source, le chemin de phéromones reste dans l'environnement et les fourmis prennent du temps pour exploiter une autre source de nourriture.

4.2 Exemple 2

Les différences avec l'exemple 1 sont :

4. la fonction de fitness est le temps pour ramener toute la nourriture au nid ;
5. le résultat de l'étude de la stochasticité est montré dans le tableau 3 pour la fitness 2 ;
6. l'évaluation de la fonction de fitness requiert environ 20 secondes pour un bon modèle et jusqu'à quelques minutes pour un très mauvais.

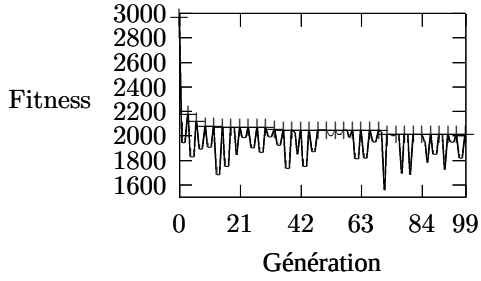


FIG. 5 – Résultat de l'algorithme génétique pour l'exemple 2

La figure 5 montre les résultats. Comme le précédent exemple, les meilleurs modèles sont ceux où les trois sources de nourriture sont exploitées en même temps. Mais le taux d'évaporation dans cet exemple est plus fort que dans l'exemple précédent. Cela améliore le comportement dynamique des fourmis : quand une source de nourriture est

épuisée, les fourmis stoppent rapidement d'aller vers cette source et en cherchent une autre.

4.3 Exemple 3 - Fitness qualitative

Les différences avec l'exemple 1 sont :

1. notre but est d'obtenir des modèles dans lesquels des lignes de fourmis peuvent émerger ;
4. la fitness est le nombre de lignes de fourmis ; pour simplifier, nous déterminons le nombre de chemins de phéromones continus entre le nid et la zone de nourriture ; le nombre de chemins est évalué tous les 10 pas de simulation et la fitness est la somme de ces valeurs durant 400 pas de simulation.
5. les résultats de l'étude de la stochasticité sont montrés dans le tableau 3 pour la fitness 3 ;
6. l'évaluation de la fonction de fitness requiert environ 30 secondes ; nous utilisons seulement un ordinateur durant une nuit.

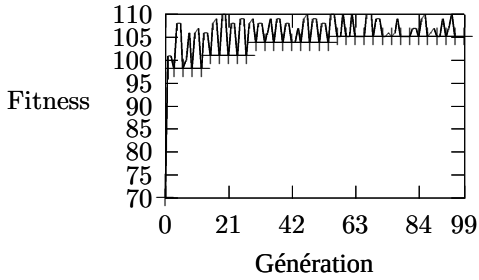


FIG. 6 – Résultat de l'algorithme génétique pour l'exemple 3

La figure 6 montre les résultats. Les meilleurs modèles exploitent encore les trois sources de nourriture en même temps. À la différence des deux précédents exemples, les taux d'évaporation et de diffusion sont très faibles (le taux d'évaporation est égal à $9,32\%$ et le taux de diffusion est égal à $1,42\%$). Cette faiblesse permet de concentrer les phéromones sur de petites zones de l'environnement sans couvrir tout l'environnement, mais aussi cela réduit la flexibilité du comportement de la colonie de fourmis.

4.4 Discussion

Comme nous l'avons déjà vu avec l'étude de la stochasticité, nous obtenons des résultats très différents en fonction du choix de la fonction de fitness. Les modèles sont fortement optimisés pour une fonction de fitness spécifique et ne peuvent pas réaliser une performance d'aussi bonne qualité avec une autre fonction de fitness. L'optimisation crée une perte de flexibilité dans la dynamique des modèles à base d'agents. Une solution possible serait d'utiliser plusieurs conditions initiales différentes pour évaluer

la fonction de fitness. Nous devons alors imaginer dans notre exemple de varier la position et la distance des sources de nourritures ou d'ajouter de nouvelles sources dynamiquement pour sélectionner des modèles qui montrent de hautes capacités d'adaptation. Cependant ceci augmenterait le temps nécessaire pour exécuter l'algorithme.

L'optimisation par l'algorithme génétique dépend aussi des contraintes imposées aux agents dans le modèle. Si un modèle a beaucoup de contraintes (peu de ressources par exemple), il est nécessaire d'optimiser le fonctionnement global. Au contraire, si les ressources sont abondantes, la pression sur le modèle pour s'adapter et optimiser son fonctionnement deviendra plus faible. Donc, l'utilisation de notre approche sera principalement bénéfique quand des contraintes sur le modèle sont fortes. Dans le cas du fourragement par une colonie de fourmis, si la fitness est le temps pour ramener toute la nourriture au nid, une importante ressource correspond au nombre de fourmis. Moins il y a de fourmis, meilleurs seront les organisations qu'elles auront besoin de créer pour fourrager de manière efficace. Au contraire, si les fourmis sont très nombreuses, le modèle sera bien performant quel qu'en soit le signalement par les phéromones. Dans certain cas, c'est alors peut-être utile de renforcer les contraintes sur le modèle pour obtenir de meilleures améliorations.

5 Conclusion

Ce papier présente une méthode, basée sur des algorithmes génétiques, pour explorer automatiquement l'espace de paramètres de modèles à base d'agents. Nous avons expliqué les difficultés spécifiques liées à l'utilisation de cette approche avec des modèles à base d'agents : le choix de la fonction de fitness, la stochasticité, le coût de calcul. Nous avons aussi montré quelques solutions possibles. Finalement nous avons suggéré une structure générale pour aider à utiliser les algorithmes génétiques dans ce contexte.

Nous avons appliqué la méthode à quelques exemples simples : le fourragement par une colonie de fourmis avec différentes fitness (à la fois quantitative et qualitative). Nous avons obtenu des modèles qui sont optimisés selon la fonction de fitness donnée, qui est choisie en relation avec des buts spécifiques du concepteur.

La prochaine étape est d'appliquer la méthode à un exemple plus complexe. Nous commençons un travail pour la simulation de la glycolyse et la phosphotransferase chez Escherichia Coli : dans ce travail, nous nous sommes intéressés au test de l'hypothèse d'hyperstructures (Amar *et al.*, 2002) : les hyperstructures sont des complexes dynamiques de molécules qui amélioreraient le comportement d'une cellule. Nous essayons de déterminer sous quelles conditions ceci peut être vrai et comment ces hyperstructures peuvent fonctionner.

Pour explorer cet exemple complexe, nous aurons besoin de développer des stratégies additionnelles pour réduire l'espace de paramètres (par l'introduction de couplages entre des paramètres), pour accélérer l'évaluation de la fonction de fitness (par le développement de méthodes d'approximation) et d'accélérer la convergence de l'algorithme génétique (par l'utilisation d'algorithme génétique interactif (Takagi, 2001)). Finalement, un autre important travail est d'explorer l'effet de la variation dynamique des conditions de simulation afin de produire des modèles plus polyvalents.

Références

AMAR P., BERNOT G. & NORRIS V. (2002). Modelling and Simulation of Large Assemblies of Proteins. *Proceedings of the Dieppe spring school on Modelling and simulation of biological processes in the context of genomics*, p. 36–42.

BEKER T. & HADANY L. (2002). Noise and elitism in evolutionary computation. In *Soft Computing Systems - Design, Management and Applications*, p. 193–203.

BEYER H.-G. (2000). Evolutionary Algorithms in Noisy Environments : Theoretical Issues and Guidelines for Practice. *Computer methods in applied mechanics and engineering*, **186**.

BRUECKNER S. & PARUNAK H. V. D. (2003). Resource-Aware Exploration of the Emergent Dynamics of Simulated Systems. *AAMAS 2003*, p. 781–788.

CANTÚ-PAZ E. & GOLDBERG D. E. (2000). Efficient parallel genetic algorithms : theory and practice. *Computer Methods in Applied Mechanics and Engineering*, **186**.

GOLDBERG D. E., DEB K. & CLARK J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex System*, **6**.

JIN Y. (2003). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing Journal*.

JIN Y., OLHOFER M. & SENDHOFF B. (2002). A Framework for Evolutionary Optimization with Approximate Fitness Functions. *IEEE Transactions on Evolutionary Computation*, **6**(5), 481–494.

TAKAGI H. (2001). Interactive evolutionary computation : fusion of the capabilities of EC optimization and human evaluation. In *Proceedings of the IEEE*, volume 89, p. 1275–1296 : IEEE Press.

TISUE S. & WILENSKY U. (2004). Netlogo : Design and Implementation of a Multi-Agent Modeling Environment. *Proceedings of Agent 2004*.

GAMA: Architecture Générique d'Agents Mobiles Adaptables

Nejla Amara-Hachmi[1] et Amal El Fallah-Seghrouchni[2]

[1] LIPN, Université Paris 13 – CNRS UMR 7030,
99 av. J. B. Clément, 93440 Villetaneuse,
na@lipn.univ-paris13.fr
[2] LIP6, Université Paris6 – CNRS UMR 7606
8 rue Capitaine Scott, 75015 Paris,
Amal.ElFallah@lip6.fr

Résumé : Les agents mobiles représentent un paradigme de programmation suscitant un intérêt croissant. Cependant, leur utilisation reste limitée vu les difficultés et surcoûts notés lors de la réalisation des systèmes d'agents adaptés aux nouveaux besoins et de mise à jour de ces systèmes opérant généralement dans des environnements complexes et évolutifs. Dans ce papier, nous proposons une architecture pour construire des agents mobiles génériques et adaptables (GAMA). L'architecture des agents GAMA est à base de composants favorisant leur modularité, leur extensibilité et leur réutilisation. Une autre caractéristique de notre proposition est de permettre l'adaptabilité des agents mobiles aux besoins des utilisateurs et à leur environnement d'exécution. Pour cela, nous augmentons l'architecture GAMA par des composants permettant de construire des agents mobiles auto-adaptables, capables de se reconfigurer dynamiquement pour s'adapter à leur contexte d'exécution, aux besoins des utilisateurs et au système multi-agents (SMA) dans lequel ils arrivent après chaque migration. Nous proposons deux types de composants, le premier est destiné à représenter et raisonner sur le contexte des agents, le second sert à effectuer les opérations de reconfiguration respectant des politiques d'adaptation.

Mots-clés : Systèmes multi-agents, agents mobiles, adaptabilité, programmation à base de composants, contexte.

1 Introduction

Dans notre travail, un agent mobile est, avant tout, un agent. Dans le domaine de l'intelligence artificielle, un agent est une entité autonome capable d'agir dans son environnement tout en communiquant avec les autres et en utilisant des compétences et des ressources propres. Un agent agit pour le compte d'un tiers (un autre agent, un utilisateur), offre des services et tend à satisfaire ses objectifs. Pour devenir mobile, un

agent se dote en plus de la propriété de migration (Fuggetta 98, Bernard et al. 02) lui permettant de se déplacer d'un site à un autre en cours d'exécution pour se rapprocher de données ou de ressources. Il se déplace avec son code et ses données propres, mais aussi avec son état d'exécution. Plusieurs avantages découlent de la capacité de migration des agents mobiles, tels que l'exécution asynchrone et autonome des tâches en déléguant cette responsabilité au niveau de l'agent plutôt que l'application, la réduction du trafic réseau en limitant les transactions distantes, et la robustesse et la tolérance aux fautes en maintenant l'exécution de l'application, même en cas de défaillance dans les supports d'exécution (machine hôte, connexion réseau).

Aujourd'hui, plusieurs plate-formes et *toolkits* pour construire et gérer des systèmes d'agents mobiles existent aussi bien dans le domaine industriel que académique. Nous citons par exemple Aglets (Lange et al. 97), D'agents (Gray et al. 98), etc. La conception et la mise en oeuvre de ces systèmes nécessite des connaissances solides de la théorie «agents et systèmes multi-agents», de bonnes compétences aussi bien en modélisation qu'en programmation, notamment en ce qui concerne le code mobile. Ceci explique les difficultés et les surcoûts constatés lors de la réalisation des systèmes d'agents mobiles adaptés aux nouvelles applications puisque les programmeurs sont quasiment toujours amenés à partir du néant pour développer de tels systèmes. De plus, les systèmes d'agents mobiles opèrent généralement dans des environnements ouverts et évolutifs nécessitant alors des opérations de mise à jour fréquentes et coûteuses. Dans ces systèmes, l'accent est souvent mis sur les plate-formes d'accueil et peu sur le concept d'agent. En effet, un agent est souvent assimilé à une collection de classes dont le contrôle est réalisé a priori par le programmeur et représentent des primitives simples du concept agent négligeant l'aspect cognitif. Tous ces constats, allant du manque de réutilisation à l'évolution continue des environnements d'exécution passant par les considérations simplistes de l'aspect agent, représentent des limites empêchant la prolifération et la vulgarisation de l'utilisation de la technologie des agents mobiles.

Dans ce travail nous proposons de remédier à ces limites en visant trois objectifs. Notre premier but est de proposer une architecture générique pour construire et modifier aisément des agents mobiles de façon à rendre la technologie plus accessible et donc plus utilisée. Notre objectif suivant est de favoriser les propriétés de modularité et de réutilisation afin d'avoir une architecture extensible et réutilisable. Enfin, nous voulons doter nos agents mobiles de la propriété d'adaptabilité pour répondre aux variations de leur environnement d'exécution et aux besoins des utilisateurs.

Pour répondre aux besoins de modularité et de réutilisation, nous avons délaissé les architectures orientées-objets traditionnelles et nous nous sommes positionnés dans la cadre de la programmation par composants (Szyperski 98) (Component-based Software Engineering). La programmation par composants est un produit naturel de l'évolution des langages orientés-objet, qui intègre en plus la capacité d'assemblage. Tandis que des objets sont exprimés au niveau du langage, les composants sont exprimés principalement en explorant leur interface publique et en favorisant la réutilisation de boîtes noires. Selon Szyperski, les composants logiciels sont des codes binaires développés indépendamment, à déploiement indépendant en univers réparti et

qui interagissent pour former une application par assemblage. Plusieurs avantages découlent de l'utilisation des composants pour le développement d'agents logiciels, notamment la possibilité de créer des agents simplement par assemblage de composants permettant ainsi leur modularité et donc leur réutilisation et leur extensibilité.

Pour mettre au point notre architecture générique d'agents mobiles, nous proposons alors un ensemble de composants offrant les services de base de l'agent ainsi que des caractéristiques spécifiques (telles que la coordination et la planification) et des fonctionnalités dépendant de l'application. Ainsi, nous renonçons à la vision classique dans laquelle un agent est une entité fixe ayant une architecture statique et nous optons pour une nouvelle orientation selon laquelle un agent est une association (à un moment donné) d'un ensemble de composants fonctionnant en coopération.

Une deuxième contribution intéressante de notre travail est de permettre l'extensibilité de l'architecture d'agent mobile proposée. En effet, l'agent mobile doit être capable d'inclure de nouvelles capacités qui n'étaient pas prévues lors de sa conception afin de prendre en compte une utilisation différente ou de nouveaux besoins "utilisateurs". Pour ce faire, l'agent est d'abord arrêté, ensuite le composant correspondant à la nouvelle fonctionnalité est greffé sur l'architecture de départ. En procédant ainsi, nous réalisons l'adaptation statique de l'architecture de nos agents mobile permettant sa flexibilité et sa réutilisabilité.

L'adaptation statique que nous exposons dans ce travail est effectuée simplement en éditant des documents XML décrivant les nouveaux composants de l'agent (pouvant être aussi des nouveaux plans ou protocoles). D'une part, cette propriété permet de mettre l'accent sur la flexibilité et les possibilités d'évolution de nos agents. D'autre part, elle permet à un utilisateur non ou peu familier avec les techniques de modélisation des agents de spécifier le comportement désiré de son agent à l'aide de documents XML. De côté de l'implémentation, ceci se justifie puisque les composants à base de documents XML peuvent être facilement traduits dans la plupart des langages de programmation pour composants, tels que EJB (Sun 00) et .NET (Microsoft 01).

Un autre apport significatif de notre travail est de permettre la reconfiguration dynamique de nos agents mobiles afin de les rendre auto-adaptatifs. Le besoin d'auto-adaptabilité des agents mobiles est motivé par leur nature les amenant à être en migration continue et par la suite à faire face à des environnements d'exécution hétérogènes. En effet, développé aux frontières du génie logiciel et des systèmes répartis, le concept d'agent mobile est *a priori* destiné à la mise en oeuvre d'applications dont les performances varient en fonction de la disponibilité et de la qualité des services et des ressources, ainsi que du volume des données échangées. Ces données peuvent varier considérablement d'une destination à une autre, c'est pourquoi l'implémentation initiale de l'agent mobile peut devenir peu convenable à l'égard de certains environnements rencontrés lors de son voyage. De plus, une adaptation statique de l'agent réalisée par l'intervention d'un utilisateur peut s'avérer, pour certaines applications, assez coûteuse. Par exemple, pour des applications de configuration de réseaux ou de commerce électronique, une adaptation statique par l'utilisateur amène à l'arrêt complet de l'application pour effectuer les modifications

requises, ce qui n'est pas sans effets néfastes dans le cas de réseaux à grande utilisation ou d'applications de commerce électronique internationales . Pour survivre, l'agent doit alors intégrer un moyen lui permettant d'ajuster son architecture aux besoins de l'environnement où il s'exécute. Cette adaptation ne sera pas réalisée simplement en entrant des sections de code spécifique mais en créant une version de l'agent répondant aux besoins du nouvel environnement par assemblage des composants appropriés. Ceci nécessite des mécanismes pour explorer l'environnement et réaliser l'ajout, l'échange ou la suppression de composant de façon dynamique sans avoir recourt à terminaison de l'agent mobile.

Dans ce papier, nous détaillons les trois volets de notre proposition, à savoir l'architecture générique d'agents mobiles à base de composants, son adaptation statique ainsi que son adaptation dynamique. Le papier est organisé de la façon suivante: dans la section 2 nous présentons l'architecture GAMA proposée. Dans la section 3, nous proposons des directives permettant d'adapter de façon statique les agents mobiles à base de composants. Enfin, dans la section 4 nous exposons une architecture permettant l'auto-adaptabilité de nos agents.

2 Architecture d'agents mobiles à base de composants

Le concept "composant" est bien adapté pour la conception des systèmes ouverts, complexes et évolutifs tels que les agents mobiles. Il a les avantages d'augmenter la productivité et baisser les coûts de réalisation des applications qui s'apprêtent à l'assemblage de composants (Szyperski 98). Les composants fournissent des services à travers des interfaces standard et se caractérisent par la propriété d'introspection qui permet de dévoiler leurs comportements et leurs données. Pour composer des composants, il suffit alors de mettre en correspondance leurs interfaces selon les services requis et fournis par chacun. Dans cette section, nous présentons une architecture générique d'agents mobiles à base de composants, que nous baptisons GAMA (Generic Adaptive Mobile Agents). Trois caractéristiques sont fondamentales pour avoir une architecture minimale d'agent qui soit en plus mobile :

- Autonomie: l'agent doit avoir la commande de ses propres processus et être responsable de ses décisions.
- Pro-activité: les agents perçoivent leur environnement et maintiennent des connaissances à son sujet. Ils doivent être capable de raisonner, éventuellement planifier leurs actions et prendre des initiatives.
- Mobilité : quand décidée, l'agent se déplace, selon un itinéraire, à travers le réseau afin d'exécuter ses tâches localement sur des sites distants.

Dans notre proposition, ces caractéristiques sont fournies par les composants suivants, représentés par des cadres épais sur la figure Fig. 1: *Contrôleur* pour l'autonomie, *Descripteur*, *Boîte au lettres*, *Actions* et *Base de connaissances* pour la pro-activité et *Mobilité* pour prendre en charge la migration de l'agent.

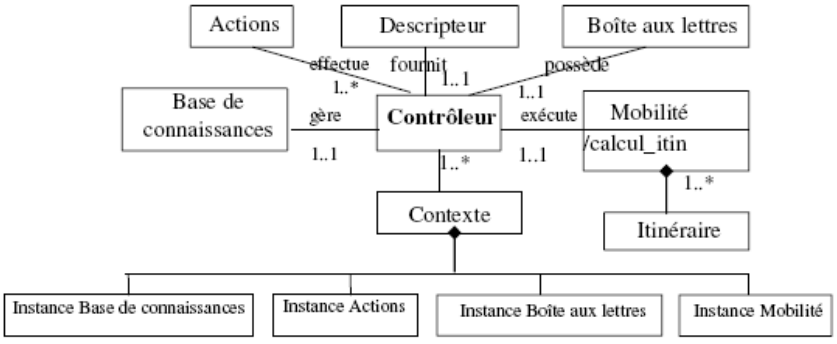


Fig. 1– Architecture générique à base de composants pour agents mobiles

2.1 Descripteur

Le composant *Descripteur* représente l'interface publique de l'agent. Il contient une description des fonctionnalités de l'agent, une liste des protocoles de communication autorisés, et le format des ACL (Agent Communication Languages) adoptés pour l'échange de messages tels que KQML (Finin et al. 94) ou FIPA-ACL (FIPA 02). Il inclue aussi une liste des destinations vers lesquelles l'agent peut migrer (son itinéraire) pour l'exécution distante de ses tâches. Dans l'état actuel du travail, les itinéraires des agents sont définis à l'avance par le concepteur de façon statique. Nous nous intéresserons à la définition des itinéraires dynamiques dans des travaux futurs.

La description des fonctionnalités de l'agent mobile contient aussi une énumération des nouvelles fonctionnalités qui peuvent être rajoutées à l'agent. Les composants décrivant ces extensions potentielles peuvent être ultérieurement greffées sur l'architecture de l'agent afin de le doter de nouvelles fonctionnalités telles que la coordination ou la planification.

2.2 Mobilité

Ce composant est responsable de la gestion de la migration de l'agent. A la réception d'une requête de migration, le composant mobilité vérifie d'abord si l'agent possède l'autorisation de migrer vers la destination spécifiée. Si l'itinéraire de l'agent est prédéfini, alors le composant mobilité le charge et affecte les sous-tâches à être exécutées sur chaque nœud de l'itinéraire. Dans certains cas, l'agent peut disposer d'une liste d'itinéraires possibles et peut choisir entre eux. Le composant mobilité doit alors comparer ces itinéraires en tenant compte des aspects de performance ou selon un certain ordre algébrique sur des itinéraires spécifiés formellement comme proposé dans (Satoh 04).

Dans le cas où l'agent doit créer son propre itinéraire, le composant mobilité calcule l'itinéraire requis pour réaliser toutes les tâches de l'agent. Par exemple, considérons une application de recherche et filtrage d'informations. Les agents

mobiles doivent migrer vers des serveurs de bases de données distants pour chercher les informations et ne garder que les résultats pertinents. Si l'agent de recherche rassemble des informations à partir d'un serveur de base de données et les rapatrie sur un autre serveur, alors l'ordre de son mouvement peut affecter le contenu des serveurs. Un tel agent doit alors se déplacer entre les serveurs selon un itinéraire spécifique. Il peut, par exemple, déterminer ses destinations en se basant sur l'information qu'il a trouvée sur le dernier serveur visité.

2.3 Boîte aux lettres

Au moyen de ce composant, l'agent communique avec toutes les entités externes. Il traite les messages entrants et sortants de/vers les autres agents du système ainsi que la plate-forme de l'agent mobile. A cause de la mobilité de nos agents, tous les messages sont traités de façon asynchrone. En effet, il n'y a pas d'échange de messages durant la migration de l'agent. Les messages entrants sont reçus lorsque l'agent est dans un état statique et sont traité de façon FIFO.

Le composant *Boîte aux lettres* est aussi responsable de la mise en forme des messages selon l'ACL spécifiée dans le composant Descripteur. Un "parseur" d'ACL est fournit comme sous-composant et peut être substitué pour supporter d'autres ACLs.

2.4 Contrôleur

Ce composant représente le noyau de notre agent mobile et réalise un couplage entre ses différents composants. Le *Contrôleur* centralise tous les événements produits par les différents composants et les achemine vers la destination adéquate. De plus, il maintient des informations sur les composants actifs dans l'agent. Cette information est représentée dans l'architecture (voir Fig.1) par l'objet *Contexte*. Durant l'exécution, l'objet *Contexte* détient les références à tous les composants actifs (représentés par les objets *Instance*) et leurs connecteurs. Les composants non utilisés peuvent être en veille jusqu'à ce qu'ils soient activés par le composant contrôleur. Une nouvelle instance du composant est alors créée et sa référence est ajoutée à l'objet *Contexte*.

2.5 Actions

Ce composant encapsule le comportement général de l'agent. Il comporte en particulier la description des tâches de chaque agent. Dans une application de recherche et filtrage d'information, un exemple d'action d'un agent mobile sur un site distant est l'interrogation locale d'une base de données.

2.6 Base de connaissances

La base de connaissances de l'agent contient toutes ses données. Elle comprend les connaissances de l'agent sur lui-même et sur les autres, ses capacités et ses buts.

3 Adaptation statique

Les progrès réalisés dans le domaine des systèmes ouverts et distribués ainsi qu'en informatique mobile ont introduit de nouvelles problématiques et créé de nouveaux besoins en terme d'adaptabilité pour la construction et la survie de ces systèmes face aux environnements d'exécution hétérogènes et évolutifs. L'adaptation de ces systèmes est motivée d'une part par les besoins de réutilisation afin de construire aisément de nouvelles applications comme illustré dans (Armor et al. 03) qui décrit une approche de développement d'agents logiciels à l'aide de composants réutilisables. D'autre part, l'adaptation est motivée par des besoins d'extensibilité pour permettre la prise en compte des conditions d'exécution particulières ainsi que les nouveaux besoins. Les systèmes qualifiés "d'adaptatifs" possèdent souvent un *manager d'adaptation* qui a la charge de contrôler le système et de configurer ses modules en tenant compte des observations recueillies lors du contrôle (Wermelinger 00). Dans le cas de l'adaptation statique, le *manager d'adaptation* est un client externe à l'application (acteur humain). L'action d'adaptabilité couvre généralement les aspects suivants:

- Modification de l'architecture d'une application par ajout/suppression de modules et la reconfiguration des relations entre eux,
- Modification de la distribution géographique d'une application (par changement de placement des composants de l'application), par exemple pour la distribution des charges,
- Modification de l'implémentation des composants,
- Modification des interfaces des composants.

Dans cette partie de notre travail, nous nous intéressons seulement au premier aspect d'adaptation des agents mobiles qui peut être entreprise par un utilisateur désirant ajouter de nouvelles fonctionnalités à l'architecture minimale d'agent proposée dans la section précédente. Nous allons illustrer cela à travers un exemple montrant que nous pouvant étendre aisément l'architecture de nos agents GAMA pour inclure de nouvelles capacités telle que la coordination.

3.1 Illustration

Pour illustrer le processus d'extension de l'architecture minimale d'un l'agent GAMA, nous proposons de le doter de la propriété de coordination en utilisant un protocole de négociation. Pour ce faire, l'utilisateur doit fournir des documents XML décrivant les propriétés et la configuration visée de l'agent. Dans notre exemple, l'utilisateur doit fournir les documents XML décrivant:

- Le composant *Coordination* (voir Fig. 2) responsable de la gestion des conversations dans lesquelles l'agent sera impliqué lors d'un processus de négociation selon le protocole choisi. Il traite les messages reçus et les événements internes pour exécuter les actions implémentées dans le composant *Actions*.

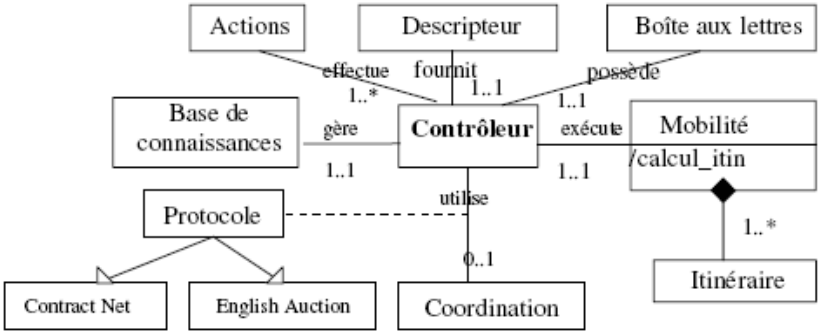


Fig. 2 – Architecture GAMA augmentée par un composant de coordination.

- Le composant *Protocole* pour coder les protocoles de coordination utilisés par l'agent. Pour chaque protocole, l'utilisateur doit fournir un schéma XML décrivant les actions à entreprendre par le composant *Coordination* afin de procéder à un processus de négociation. En effet, la description XML de chaque protocole consiste en la description des messages échangés et la description des rôles participant à la négociation. Pour qu'un agent puisse utiliser un nouveau protocole de négociation, il suffit qu'il ait à sa disposition la description XML correspondante (Armor et al. 03). Dans le cadre de notre exemple, nous présentons une illustration possible du composant *Protocole* décrivant une négociation selon le protocole *Contract Net* (Smith 80). Selon ce protocole, la négociation est initiée par un agent qui émet un appel d'offres décrivant une tâche et les conditions nécessaires à son exécution. Les agents recevant l'appel d'offres tentent à satisfaire la demande de l'initiateur en lui faisant des propositions qui tiennent aussi compte de leur profit personnel. Le processus continue jusqu'à ce qu'un accord soit établi. Ce processus peut être décrit parfaitement par un schéma XML.

Pour intégrer le composant *Coordination* dans l'architecture de l'agent, il suffit d'inclure les informations le décrivant dans le document XML de déploiement du composant *Contrôleur* comme décrit dans le code XML suivant :

```
<?xml version="1.0" ">
<AgentDescription xmlns:xsi="http://www.example/XMLSchema"Location=
"C:\coordinating\xml\AgentDescription.xsd">
<AgentInterfaceComponent>
      <InterfaceDescription href="http://…/Interface.xml" notation="string"/>
      <DeploymentInfo href=" http://…/InterfaceDeployment.xml" notation="String"/>
</ AgentInterfaceComponent >
<MobilityComponent>
      <InterfaceDescription href="http://…/Mobility.xml" notation="String"/>
     <DeploymentInfo href=" http://…/MobilityDeployment.xml" notation="String"/>
</MobilityComponent>
<BasicActionsComponent>
      <InterfaceDescription href="http://…/Actions.xml" notation="String"/>
      <DeploymentInfo href=" http://…/ActionsDeployment.xml" notation="String"/>
</BasicActionsComponent>
<MessagesCentreComponent>
      <ACLRepresentation ACLParser="http://…/ACLString.class" format="String"/>
      </MessagesCentreComponent>
<KnowledgeBaseContent>
```

```
      <AcquaintanceDatabase resource="">
</KnowledgeBaseContent>
<CoordinationComponent>
     <InterfaceDescription href="http://…/Coordination.xml" notation="String"/>
     <DeploymentInfo href=" http://…/ Coordination xml" notation="String"/>
</CoordinationComponent>
<ProtocolsComponent>
     <ProtocolDescription href="http://…/protocol/ContractNet.xml"/>
</ ProtocolsComponent >
```

Le comportement initial doit aussi être en suite spécifié pour permettre à l'agent de commencer son exécution.

4 Adaptation dynamique

Idéalement, les développeurs d'agents mobiles devraient être capables de se concentrer sur le code fonctionnel de leurs applications sans avoir à se soucier des caractéristiques et des ressources disponibles sur les plate-formes sur lesquelles leurs agents vont s'exécuter. Dans ce travail, nous nous sommes alors fixé l'objectif de mettre au point une solution à ce problème en proposant une architecture d'agents mobiles auto-adaptables. Pour cela, nous nous somme inspirés des travaux réalisés dans le domaine de programmation par composants (Maes 87, Cazzola et al. 97) pour doter l'architecture générique d'agent mobile (présentée dans la section 2) de politiques d'adaptation. Ces politiques, qui prennent la forme de règles d'adaptation, viennent enrichir le noyau générique de l'agent pour lui conférer la capacité de se reconfigurer dynamiquement (sans l'intervention de l'utilisateur) afin de s'auto-adapter aux nouveaux environnements d'exécution. Dans cette section, nous présentons les principes généraux de l'adaptation dynamique, ensuite nous exposons les composants que nous proposons pour permettre l'auto-adaptabilité de nos agents GAMA.

4.1 Principes de l'adaptation dynamique

De manière générale, adapter signifie: *rendre (un dispositif, des mesures, etc.) apte à assurer ses fonctions dans des conditions particulières ou nouvelles*. L'auto-adaptation d'un système logiciel le rend capable d'intégrer des modifications de manière transparente à ses utilisateurs. Ces modifications peuvent être motivées par différents besoins tels que l'optimisation des performances, le masquage des pannes, les évolutions techniques et les évolutions fonctionnelles. Dans ce travail, nous considérons uniquement l'adaptation des agents mobiles aux évolutions fonctionnelles.

D'après la synthèse faite dans (Yahiaoui 04), deux types d'adaptation dynamique existent : fonctionnelle et technique. L'adaptation fonctionnelle consiste en la modification des services fournis par l'application, c'est à dire par ajout de nouveaux services ou modification des services existants. L'adaptation technique concerne la manière dont les services d'une application sont fournis, par ajout ou suppression de certains aspects techniques tels que la persistance ou la sécurité. Dans ce travail, nous nous intéressons seulement à l'adaptation fonctionnelle puisque nous visons l'adaptation des services fournis par un agent mobile. Nous considérons également que l'adaptation dynamique permet aux agents mobiles de se reconfigurer dynamiquement

pour répondre aux évolutions de leurs contextes d'exécution. Pour ce faire, un processus de trois étapes est requis:

- Une première étape d'observation est nécessaire pour contrôler et évaluer l'application et son environnement,
- Une deuxième étape de décision qui détermine les modifications à entreprendre selon les résultats de l'observation,
- Une troisième étape d'exécution ayant pour objectif de mettre en œuvre la décision prise précédemment.

Dans la section suivante, nous proposons d'augmenter l'architecture générique de nos agents mobiles par des nouveaux composants leur permettant d'observer leurs contextes d'exécution et raisonner sur les évènements susceptibles de déclencher des opérations de reconfiguration dynamique qu'ils réaliseront par la suite.

4.2 Des composants pour la reconfiguration dynamique

Les agents mobiles que nous construisons selon l'architecture proposée dans la section 2, migrent à travers le réseau d'hôte à hôte. A chaque destination l'agent mobile est reçu au moyen d'une plate-forme d'accueil installée sur toutes les machines. Une plate-forme d'agents mobiles fournit des services permettant de gérer la mobilité, le cycle de vie, la persistance et la sécurité des agents qu'elle héberge. Elle fournit aussi certains services communs tels que la recherche d'agents et le registre d'agents; et réalise le lien entre les agents mobiles et le système d'exploitation. Dans ce travail, nous considérons que le contexte d'exécution d'un agent mobile est reflété au niveau de sa plate-forme d'accueil. En effet, nous distinguons trois éléments décrivant le contexte d'un agent mobile:

- Le contexte physique regroupant les caractéristiques des ressources physiques mise à la disposition de l'agent, telles que la taille de la mémoire de la machine et la puissance de son processeur, la capacité et la disponibilité de la bande passante du réseau.
- Le contexte social de l'agent décrivant le système multi-agents (SMA) local dans lequel l'agent mobile arrive. Ce contexte social regroupe notamment les protocoles de communication[1] (par exemple, KQML (Finin 94) ou FIPA-ACL (FIPA 02)) et de coordination[2] (par exemple, Contract Net (Smith 80) ou English Auction (FIPA 00)) utilisés dans le SMA destination.
- Le profil de l'utilisateur (propriétaire de l'agent) qui peut, par exemple, avoir des préférences telles que imposer des restrictions sur les fichiers ou les données manipulées par l'agent.

[1] La communication est l'échange de flux d'information entre l'agent et son environnement (incluant les autres agents).

[2] Deux agents A et B sont en interaction (Van Dyke Parunak et al. 02) si les conséquences des actions de B sur son environnement influencent l'exécution des actions de A. La coordination entre agents est vue comme une forme plus élaborée d'interaction associée à de la communication.

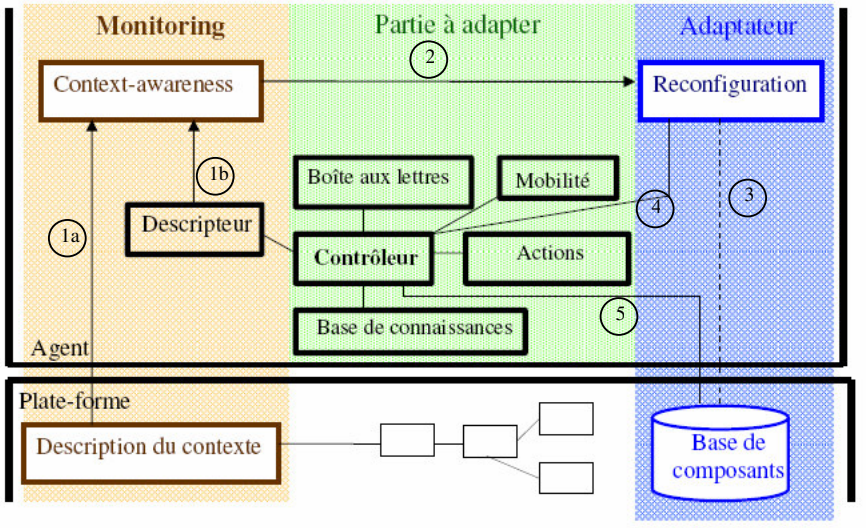


Fig. 3– Architecture d'agent mobile auto-adaptable.

Nous proposons alors de fournir une description de ces éléments par l'intermédiaire d'un nouveau composant *Description du contexte* que nous intégrons au niveau des plates-formes d'accueil des agents mobiles (voir Fig. 3). Suite à chaque migration, ce composant fournit à l'agent un moyen d'investigation des nouveaux environnements d'exécution. L'agent pourra par la suite raisonner sur ce nouveau contexte et décider s'il y a besoin d'auto-adaptation. Pour pouvoir conduire ce raisonnement, nous avons opté pour une représentation formalisée du contexte d'exécution d'un agent mobile, mettant en évidence les attributs caractérisant ce contexte ainsi que leurs valeurs. Nous avons alors adapté la recommandation CC/PP *Composite Capabilities/Preferences Profiles* (Profils composites de capacités/préférences) du W3C (W3C 04) à notre besoin. Cette recommandation nous permet de décrire chaque élément du contexte d'un agent à l'aide de son profil. Un profil CC/PP est construit selon une hiérarchie de deux niveaux où chaque profil possède au moins un composant et chaque composant possède au moins un attribut.

Dans la figure Fig.4, nous présentons la description du profil du contexte d'un agent mobile comme décrit ci-dessus. La description du contexte d'exécution d'un agent mobile, ne représente que l'entrée d'un processus de reconfiguration dynamique. Il faut par la suite que l'agent puisse raisonner sur ce contexte et le comparer à ce qui est fournit par ses composants. En cas où une auto-adaptation est sollicitée, l'agent doit pouvoir retrouver les composants adéquats à intégrer selon une certaine politique d'adaptation. Pour ce faire, nous proposons d'enrichir notre architecture d'agent mobile par deux nouveaux composants : *Context-awareness* pour

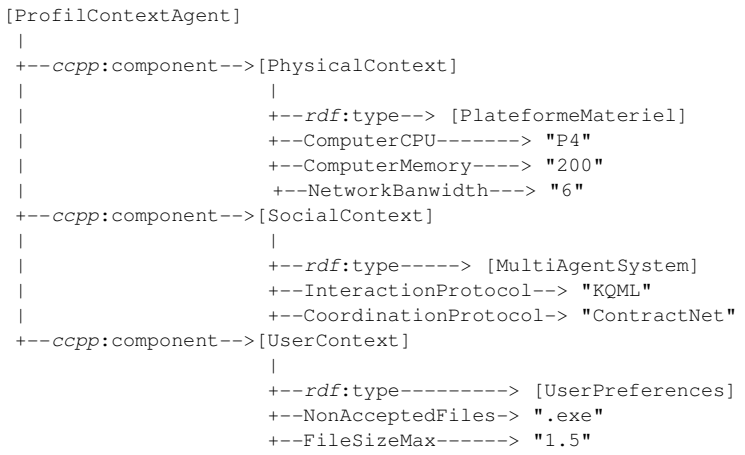


Fig. 4 – Représentation du contexte d'un agent mobile à l'aide d'un profil CC/PP.

le raisonnement sur le contexte d'exécution et *Reconfiguration* pour la mise en œuvre de l'adaptation dynamique (voir Fig. 3).

Après chaque migration, le composant *Context-awareness* d'un agent mobile se réfère au composant *Description du contexte* sur la plate-forme de la machine hôte. Il vérifie la compatibilité de la configuration actuelle de l'agent (exhibée à travers le composant *Descripteur*) avec les profils du nouvel environnement (voir liens 1a et 1b de Fig. 3). Si le résultat de la comparaison est négatif, un évènement est généré pour notifier le composant *Reconfiguration* en vue d'entreprendre une adaptation (voir lien 2 de Fig3). La comparaison peut se faire par évaluation d'expressions booléennes codées dans le composant *Context-awareness.* Par exemple, la règle suivante permet de comparer le protocole de coordination utilisé par l'agent et celui utilisé par le système multi-agents qui va l'accueillir (son contexte social) :

```
si ┐(ProfilContextAgent.SocialContext.CoordinationProtocol
==  AgentDescripteur.CoordinationMetaModel.Name)
alors Notify(Reconfiguration)
```

Une fois la reconfiguration sollicitée, elle doit se faire sans terminaison de l'agent. Le composant *Reconfiguration* applique alors une politique d'adaptation lui permettant de déterminer les composants à ajouter, à supprimer ou à remplacer dans l'architecture de l'agent, disponible dans une base de composants présente sur la plate-forme d'accueil (voir lien 3 de Fig. 3) puis avise le Contrôleur (voir lien 4 de Fig. 3). Ces composants sont ensuite intégrés (dans les cas d'ajout et de remplacement) à l'architecture de l'agent par le composant *Contrôleur* (voir lien 5 de Fig.) qui réalise la reconfiguration de l'agent. En continuant sur l'exemple précédent, le composant *CoordinationProtocol* de l'agent mobile n'est pas compatible avec le profil du système multi-agents local, il est alors remplacé par une composant représentant un autre protocole selon l'algorithme suivant :

```
Algorithme remplacement (Coord1, Coord2)
    Verrouiller les canaux de communication de et vers Coord1 pour
    éviter de perdre les messages échangés;
    Mettre Coord1 à l'état « passif » par appel à la méthode
    passivate;
    Transférer l'état de Coord1 vers Coord2 (transfert des
    propriétés publiques) ;
    Déconnexion de Coord1 et Connexion de Coord2;
    Mettre Coord2 à l'état « actif » par appel à la méthode
    activate.
```

Après le remplacement, le *Descripteur* de l'agent est mis à jour en remplaçant l'entrée correspondante au composant *Coord1* par une autre correspondant à *Coord2*.

5 Conclusion

Dans ce papier, nous avons proposé une architecture d'agent mobile générique, réutilisable et extensible baptisée GAMA. Nous avons également proposé deux manières d'adapter l'architecture d'agents GAMA aux variations de leurs environnements d'exécution. La première est une adaptation statique effectuée par l'intervention d'un acteur humain réalise l'adaptation en agissant sur les documents de déploiement des composants. Le second type d'adaptation que nous avons proposé est dynamique et se fait de manière transparente aux utilisateurs. Pour ce faire, nous avons augmenté l'architecture des agents GAMA par des composants permettant l'observation et le raisonnement sur les contextes d'exécution de l'agent. Ensuite, un autre composant se charge d'effectuer la reconfiguration décidée selon certaines politiques d'adaptation. Concernant la représentation de contexte, nous avons utilisés les profils CC/PP du W3C. Les règles de politiques d'adaptation d'agents mobiles sont en cours d'étude ainsi que les mécanismes de reconfiguration et de recherche de composants. Une modélisation des états des composants d'un agent mobile ainsi que des algorithmes élaborés de transfert d'états seront abordés dans un prochain travail.

Un prototype de l'agent GAMA et sa plate-forme d'accueil ont été développés en EJB. Dans la suite, nous projetons de finaliser l'implémentation des deux types d'adaptation proposés et de travailler sur la réorganization dynamique des systèmes d'agents mobiles.

Références

Amor M., L. Fuentes, L. Mandow, J. M. Troya (2003). "Building Software Agents from Software Components". CAEPIA 2003: 221-230

Bellissard L., N. De Palma and M. Riveill (1998). "Dynamic reconfiguration of agent-based applications", ACM European SIGOPS Workshop, Sintra (Portugal), September 7th-10th 1998.

Bernard G., Ismail L. (2002). "Apport des agents mobiles à l'exécution répartie". Revue des Sciences et Technologies de l'Information, série Technique et Science Informatiques, vol. 21, n° 6, 2002, p. 771-796, Hermès Science Publications, Lavoisier.

Brazier, F.M.T., Jonker, C.M., Treur, J. (2004). "Principles of Component-Based Design of Intelligent Agents". Data and Knowledge Engineering 41 (2002) 1-28

Cazzola W., A.Sosio, A.Savigni, F.Tisato.(1999). "Architectural Reflection: Concepts, Design, and Evaluation". Technical Report RI-DSI 234-99, DSI. University degli Studi di Milano. May 1999.

Finin T., Fritzson R., McKay D and McEntire R. (1994). "KQML as an Agent Communication Language". Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94).

FIPA (2002), "FIPA ACL Message Structure Specification", available at: http://www.fipa.org/specs/fipa00061/SC00061G.html

Fuggetta A., Picco G. P. et Vigna G. (1998). ″Understanding code mobility″. IEEE Transactions on Software Engineering, Vol 24, No 5, pp 342-361.

Gray R. S., Kotz D., Cybenko G., and Rus D. (1998). ″D'agents : Security in a multiple-language, mobile-agent system″, Mobile Agents and Security, LNCS, vol. 1419, pp. 154-187.

Herlea, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E. (1999). "Specification of Behavioural Requirements within Compositional Multi-Agent System Design". In: F.J. Garijo, M. Boman (eds.), Multi-Agent System Engineering, Proc. of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'99. Lecture Notes in AI, vol. 1647, Springer Verlag, pp. 8-27.

Kickzales G., J.Lamping, A.Mendhekar, C.Maeda, C.Lopes, J-M.Loingtier, J.Irwin.(1997). "Aspect-Oriented Programming". Proceedings of the ECOOP'97 Conference. June1997. Finland.

Lange D. B., Oshima M., Karjoth M., Kosaka K. (1997). "Aglets: Programming Mobile Agents in Java". Proc. of the Worldwide Computing and Its Applications, International Conference, WWCA '97,Lecture Notes in Computer Science, vol. 1274, Springer Verlag, pp. 253-266.

Maes P. (1987). "Concepts and experiments in computation reflection". ACM SIGPLAN Notices, 22(12):147-155.

Mathieu P. and M.H. Verrons (2003). "A generic negotiation model for MAS using XML," International Workshop series agents for Business Automation: Research and development, pp. 4262-4267.

Microsoft Corporation (2001). "Microsoft .NET Framework". http://java.sun.com/products/ejb/

Satoh I., "Selection of mobile agents," th International Conference on Distributed Computing Systems (ICDCS 2004), 24-26 March 2004, Hachioji, Tokyo, Japan. IEEE Computer Society 2004, pp. 484-493.

Silva A. and Delgado J (1998) "The agent pattern for mobile agent systems," 3rd European Conference on Pattern Languages for Programming end Computing..

Smith R. G.(1980). "The Contract Net Protocol : highlevel communication and control in a distributed problem solver". IEEE Transactions on computers, C-29(12), 1104–1113, December 1980.

Sun Microsystems Inc. (2000). "Entreprise JavaBeans Specifications". Janvier 2000. http://java.sun.com/products/ejb/

Szyperski C. (1998). "Component Software. Beyond Object-Oriented programming,". Addison-Wesley, Reading Mass 1998.

Pawlak R., L. Seinturier, L. Duchien, G. Florin. (2001)."JAC: A Flexible Solution for Aspect-Oriented Programming in Java". In Refection 2001. September 2001. Japan.

Yahiaoui N., B. Traverson and N. Levy (2004). "Classification and comparison of adaptable platforms". First International Workshop on Coordination and Adaptation Techniques for Software WCAT 04, June 2004, Oslo, Norway.

Van Dyke Parunak H., Brueckner S., Fleisher M. et Odell J. (2002). "Co-X: defining what agents do together". In AAMAS'02 Workshop: Teamwork and coalition formation. July 2002. Bologna, Italy.

Résolution d'emploi du temps dynamique et distribuée par auto-organisation coopérative

Gauthier Picard

IRIT - Université Paul Sabatier - Toulouse III
118, route de Narbonne
31062 Toulouse Cedex, France
picard@irit.fr

Résumé : La coopération est un moyen pour les systèmes multi-agents de fonctionner plus efficacement et de manière plus adaptative. Elle peut être vue comme un critère local de réorganisation pour les agents afin de produire une fonction globale et collective plus adaptée. Cet article montre une application des comportements coopératifs à un problème de résolution d'emploi du temps, ETTO, dans lequel la satisfaction de contrainte est distribuée dans des agents coopératifs. Cette application a été prototypée et montre des résultats positifs en terme d'adaptation, de robustesse et d'efficacité de cette approche.

Mots-clés : Systèmes multi-agents, auto-organisation, coopération

1 Introduction

En réponse à la complexité grandissante des environnements logiciels –en nombre de participants ou en dynamique– les systèmes artificiels sont de plus en plus difficiles à concevoir convenablement. La fonction globale de ces systèmes est souvent spécifiée incomplètement ou de manière floue, mais les parties restent facilement identifiables et les théories de micro-niveau les caractérisant sont connues. Des tels phénomènes, qualifiés d'*émergents*, sont étudiés par les biologistes et les physiciens depuis des années. Les deux principales propriétés de ces systèmes sont l'*irréductibilité* des macro-théories aux micro-théories (Ali *et al.*, 1997) et les *mécanismes auto-organisateurs* qui sont à l'origine de l'adaptation et de l'apparition de nouvelles propriétés émergentes (Goldstein, 1999).

Les cartes de Kohonen ou les algorithmes fourmis sont deux exemple pertinents de transcriptions de mécanismes auto-organisateurs (Kohonen, 2001; Bonabeau *et al.*, 1997). Pour être appliqués à des tâches moins spécifiques, les mécanismes nécessitent de fournir aux parties des capacités cognitives afin de décider quand se réorganiser pour s'adapter à la pression de l'environnement et atteindre le but global. Les parties deviennent alors des agents. En réponse à ce besoin de prise de décision, l'approche par *systèmes multi-agents adaptatifs* (ou *AMAS*) propose la *coopération* comme critère local –pour les parties ou agents– de réorganisation. Ici, la coopération n'est pas

limitée au partage de ressources ou de tâches, mais est un directive comportementale. De plus, la coopération est vue de manière proscriptive : les agents doivent localement changer leurs interactions lorsqu'il sont en *situation non coopérative* (ou *NCS*). Ces changements locaux peuvent être vus, au niveau global, comme une réorganisation du système. Dans un AMAS, un agent est coopératif s'il vérifie les trois méta-règles suivantes (Camps *et al.*, 1999) :

c_{per} : les signaux perçus sont compris sans ambiguïté ;
c_{dec} : les informations reçues produisent un raisonnement ;
c_{act} : le raisonnement produit une action utile à autrui.

Si un agent détecte qu'il est en NCS ($\neg c_{per} \vee \neg c_{dec} \vee \neg c_{act}$), il doit agir pour revenir à un état coopératif. Le théorème de l'adéquation fonctionnelle (Georgé *et al.*, 2004) assure que le système produit une fonction globale correcte –avec aucune interaction inutile ou antinomique avec son environnement– si tous les agents sont coopératifs.

Par conséquent, concevoir des systèmes adaptatifs revient à fournir des agents coopératifs et ainsi assurer l'adéquation fonctionnelle du système. Dans les sections suivantes, cette approche est illustrée en définissant des comportements coopératifs pour des agents devant résoudre dynamiquement un problème d'emploi du temps universitaire. Les enseignants et les groupes d'étudiants doivent trouver des partenaires, des créneaux horaires et des salles pour donner ou recevoir des enseignements. Chaque acteur possède des contraintes concernant ces disponibilités ou des équipements nécessaires. De plus, un enseignant peut ajouter ou retirer des contraintes à n'importe quel moment de la résolution via une interface adaptée. Un telle application nécessite clairement de l'adaptation et de la robustesse. Le système doit être capable de s'adapter aux perturbations environnementales (modifications de contraintes) et ne pas calculer de nouvelles solutions, depuis le début, à chaque changement. L'organisation collective adéquate doit émerger des interactions locales entre acteurs.

Le système de résolution a été nommé ETTO, pour *Emergent TimeTabling Organization*. Deux types d'agents ont été identifiés et sont présentés dans la section 2. Ces agents respectent plusieurs règles de coopération qui sont exposées dans la section 3. Des expérimentations montrent des résultats sur l'adaptation et la robustesse de cette approche dans la section 4.

2 Les agents dans ETTO

Deux différentes classes d'agents ont été identifiées dans ETTO : les *Representative Agents* (RA) et les *Booking Agents* (BA). Les RAs délèguent l'exploration de l'espace des solutions (une grille à n dimension de cellules représentant le planning) aux BAs. Chaque cellule c_i de la grille est contrainte (créneau, nombre de places, etc), ce qui est regroupé dans un ensemble $C(c_i)$. La coopération entre les agents doit mener à une organisation correcte en explorant efficacement la grille.

2.1 Representative Agents

Les RAs forment l'interface entre les acteurs humains (enseignants ou étudiants) et le système de résolution d'emploi du temps. Ils possèdent des contraintes (qualifiées

d'*intrinsèques*) concernant la disponibilité, les nécessités d'équipements (projecteurs, tableaux blancs, etc) ou tout autre type de contrainte personnelle. Pour explorer efficacement les possibilités de partenariat et de réservation de salles, ces agents délèguent l'exploration aux BAs. Un RA (appelé *proxy*) créé autant de BAs (appelés *délégués*) qu'il a de cours à donner ou à recevoir. Il place initialement ces BAs de manière aléatoire dans la grille du planning. Les BAs d'un même RA sont appelés *frères*. La cohérence entre des délégués frères est assurée par leur proxy.

La tâche d'un RA est simple : prévenir ses BAs délégués lorsque l'utilisateur ajoute ou supprime des contraintes et d'informer tous ses BAs délégués lorsqu'un de ses BAs délégués produit une nouvelle contrainte (dite *induite*) suite à une réservation, par exemple, ce qui doit inciter ses frères à ne pas réserver au même créneau horaire.

2.2 Booking Agents

Les BAs sont les véritables acteurs de l'auto-organisation dans ETTO. Ils doivent réserver les créneaux et les salles et trouver des partenaires (étudiants pour les enseignants et *vice versa*) en accord avec les contraintes de leur proxy.

En situation coopérative, un BA, qui est dans une cellule de la grille (i.e. un créneau horaire dans une salle pour un jour donné) la réserve et établit un partenariat avec un autre BA. Mais cette situation nominale n'est pas assurée au commencement car les BAs sont positionnés aléatoirement. Les BAs ont alors besoin de se réorganiser dans la grille en changeant leurs partenariats et réservations, pour atteindre un emploi du temps qui leur paraît adéquat. De telles situations sont des NCS. Par conséquent, les BAs doivent être capables de répondre à ces situations en respectant des règles de coopération (voir section 3).

Les actions qu'un BA peut effectuer sont simples : établir (ou annuler) un partenariat avec un autre BA, réserver (ou libérer) une cellule (en marquant ou supprimant un marqueur avec son adresse), se déplacer dans une autre cellule et envoyer des messages aux autres agents qu'il *connaît*. Un BA ne connaît que son proxy et les BAs qu'il rencontre dans les cellules, en cours de résolution.

Le cycle de vie d'un BA est un cycle classique "perception-décision-action" comme proposé dans (Capera *et al.*, 2003) :

1. Durant la phase de *perception*, le BA vérifie les messages reçus (des autres BAs ou de son proxy) et met à jour les données concernant la cellule qu'il occupe (les BAs dans la cellule, les marqueurs, les propriétés de la cellule) ;
2. Durant la phase de *décision*, le BA doit choisir une action à exécuter pour être le plus coopératif possible, en accord avec les règles de coopération ;
3. Durant la phase d'*action*, le BA exécute l'action choisie.

Pour effectuer ses tâches, un BA possède les propriétés, capacités et connaissances *locales* suivantes :

- sa position courante dans la grille ($cell(ba_i)$) ;
- son partenaire courant ($partnership(ba_i, ba_j)$ avec $i \neq j$) ;
- sa réservation courante ($reservation(ba_i, c_j)$) ;
- son proxy ($proxy(ba_i)$) ;

- son temps de recherche ($time(ba_i)$) pour une réservation ;
- le créneau horaire correspondant à une cellule ($slot(c)$) ;
- une mémoire limitée de BAs connus ($knows(ba_i, ba_j)$) pour leur envoyer des messages, qui est vide au début de la résolution et qui se met à jour au cours de l'exploration de la grille ;
- un ensemble de contraintes intrinsèques (CI_{ba_i}) hérité du proxy à la création du BA ;
- un ensemble de contraintes induites par ses frères (CB_{ba_i}) qui sont attachées et mises à jours par le proxy lorsqu'un de ses frères réserve une cellule pour éviter les situations d'ubiquité (deux BAs d'un même RA réservent deux cellules correspondant à un même créneau horaire, par exemple) ;
- un ensemble de contraintes induites par son partenaire (CP_{ba_i}) qui sont créées et mises à jour à chaque partenariat pour prendre en compte les préférences du partenaire ;
- un ensemble de contraintes induites par sa réservation (CR_{ba_i}) pour éviter des partenariats avec des BAs non disponibles pour la réservation ;
- un ensemble de contraintes d'un premier ensemble qui sont non compatibles avec les contraintes d'un second ensemble ($nonCompatible(C_i, C_j) \subseteq C_i$) ;
- une fonction de pondération des contraintes ($w(c_i) > 0$). Plus le poids est élevé, plus la contrainte est difficile à relâcher. En conséquence, une contrainte est non relâchable si $w(c_i) = +\infty$.

Nous définissons la macro NC afin de simplifier les notations futures :

Définition 1
L'ensemble des contraintes non compatibles entre deux BAs est $NC_{ba_i,ba_j} = nonCompatible(CI_{ba_i} \cup CB_{ba_i} \cup CR_{ba_i}, CI_{ba_j} \cup CB_{ba_j} \cup CR_{ba_j})$.

Pour déterminer les contraintes non compatibles entre deux BAs, les contraintes provenant des partenaires (CP) ne sont pas prises en compte. De la même manière, pour déterminer si une cellule est compatible avec les contraintes d'un BA, les contraintes de la réservation courante du BA ne sont pas incluses :

Définition 2
L'ensemble des contraintes non compatibles entre un BA et une cellule est $NC_{ba_i,c_j} = nonCompatible(CI_{ba_i} \cup CB_{ba_i} \cup CP_{ba_i}, C_{c_j})$.

En utilisant NC, deux porteurs de contraintes (BA ou cellule) peuvent savoir s'ils sont compatibles :

Définition 3
$compatible(x, y) \equiv (NC_{x,y} = \emptyset)$.

Avant de commencer la résolution, il n'y a pas de moyen absolu de décider quels sont les sous-problèmes les plus difficiles à résoudre. De plus, le degré de difficulté peut évoluer à cause de la dynamique induite par l'environnement. Par conséquent, durant le processus de résolution, chaque agent doit être capable d'évaluer la difficulté qu'il a à trouver un partenaire ou une réservation. Un BA ba_i peut calculer le coût d'une

réservation d'une cellule c_j ($rCost(ba_i, c_j)$) et le coût d'un partenariat avec un autre BA ba_j ($pCost(ba_i, ba_j)$) comme suit :

- $rCost(ba_i, c_j) = (\sum_{c \in NC_{ba_i, c_j}} w(c))/time(ba_i)$;
- $pCost(ba_i, ba_j) = \sum_{c \in NC_{ba_i, ba_j}} w(c)$;

Diviser par le temps de recherche, $time(ba_i)$, permet de définir le BA le plus prioritaire : celui qui cherche une cellule depuis le plus longtemps. En effet, informellement, il semble plus coopératif de privilégier les agents ayant du mal à trouver une place dans l'organisation.

2.3 Comportement basique

Les BAs ont deux buts orthogonaux : *trouver un partenaire* et *trouver une réservation*. L'algorithme principal de résolution est distribué dans les BAs et repose sur la coopération entre les agents. La résolution est le résultat des interactions dynamiques entre les entités distribuées (BAs). Comme les BAs doivent atteindre deux buts individuels, le comportement nominal peut être exprimé en termes d'atteinte de ces buts :

Algorithme 1 – Comportement nominal pour un BA.

```
while alive do
  processMessages()
  if partner AND reservation then
    if reservation is optimal then
      moveTo(reservedCell)
    else
      processCurrentCell()
    endif
  else
    moveTo(nextCell);
    addBAsToMemory();
    processEncounteredBAs();
    if NOT (reservation OR partner) then
      processCurrentCell()
    endif
  endif
done
```

Durant la phase de perception, le BA vérifie sa boîte aux lettres, dans laquelle les autres BAs déposent des messages à propos de partenariats ou de réservations. Si le BA a atteint son but (partenariat et réservation), en conséquence des messages reçus, il se déplace jusqu'à sa cellule réservée seulement si sa réservation n'est pas contrainte. Dans le cas où il a relâché des contraintes, il continuera à explorer la grille pour trouver une meilleure solution. Si le BA manque de partenaire ou de réservation, il explore la grille et analyse les BAs rencontrés en mémoire et les cellules connues, i.e. il vérifie si les BAs rencontrés ou les cellules visitées pourraient satisfaire ses propres contraintes.

2.4 Gestion des contraintes et travaux relatifs

Les actions peuvent mener à ajouter des contraintes induites. Par exemple, un BA qui réserve une cellule correspondant à un créneau horaire pour un jour donné avertit ses frères, via son proxy, que ce créneau horaire est inaccessible pour éviter les situations d'ubiquité. A contrario, si un BA annule une réservation, il doit en informer ses frères. Ainsi, un BA doit manipuler deux types de contraintes : les intrinsèques, provenant de l'acteur représenté par son RA proxy, et les induites, provenant de ses frères, de son partenaire et de sa réservation. Bien sûr, certains problèmes n'ont pas de solution sans relaxation de contraintes. En conséquence, les BAs doivent être capables d'affecter des priorités aux contraintes, comme dans les CSPs pondérés ou flous (Bistarelli *et al.*, 1999). Mais contrairement aux CSPs dynamiques classiques (Dechter *et al.*, 1991), la mémoire des états précédents est distribuée parmi les BAs. Enfin, contrairement à toutes ces approches, les BAs raisonnent uniquement sur un nombre limité de BAs connus pour trouver une bonne solution comme dans les CPSs distribués (Yokoo *et al.*, 1998). Comme les BAs sont des agents, ils n'ont pas de connaissance globale. Ainsi, la satisfaction de contraintes est partagée par les BAs, et la solution émerge de leurs interactions point-à-point locales. Cependant, notre approche reste différente des approches susmentionnées, car l'objectif principal n'est pas de fournir un algorithme complet et correct, mais de proposer des mécanismes locaux et robustes, capables de mettre en oeuvre une résolution globale. De manière similaire aux approches par algorithmes fourmis pour résoudre les problèmes d'emploi du temps (Socha *et al.*, 2002), les BAs altèrent leur environnement (la grille) grâce à des marqueurs pour indiquer la cellule réservée et pour contraindre les autres agents. La principale différence avec l'usage de phéromones est la façon dont les marqueurs disparaissent. Chez les fourmis artificielles, les marqueurs (phéromones) s'évaporent avec le temps avec une vitesse difficile à déterminer pour le concepteur. Dans notre algorithme, les marqueurs sont supprimés par des critères déterminismtes à la suite d'une négociation entre des BAs lors d'un conflit de réservation (voir section 3.4), par exemple.

3 Règles d'auto-organisation coopérative

L'algorithme de résolution distribuée de CSP que nous proposons est distribué dans les BAs et repose sur la coopération. Comme précédemment dit dans la section 2, le comportement nominal d'un agent n'est pas suffisant pour mener le collectif vers une organisation adéquate. Les BAs doivent respecter des règles de coopération pour atteindre un état global correct[1]. Concevoir des agents coopératifs revient à implémenter les méta-règles de coopération (voir section 1). Cinq situations pour la réorganisation sont identifiées. Les deux premières sont des situations qui ne respectent pas la méta-règle c_{dec}. Les trois suivantes ne respectent pas c_{dec} Dans l'exemple d'ETTO, comme nous ne nous focalisons que sur les BAs, ils n'y a pas de violation de la règle c_{per} car tous les agents sont identiques et peuvent se comprendre.

L'idée est de concevoir ces règles comme des *exceptions* en programmation objet

[1] i.e. le système produit une fonction sans composante antinomique ou inutile pour son environnement.

classique, au niveau des agents et non au niveau des instructions. Ce concept convient parfaitement à l'approche proscriptive proposée dans (Capera *et al.*, 2003). Comme pour les exceptions, les concepteurs doivent spécifier la condition de déclenchement et l'action à effectuer en retour. Les règles suivantes de coopération sont ainsi présentées comme des paires condition-action. Les conditions ne sont pas forcément exclusives. Cependant, il faut prendre soin de définir une politique de choix des actions en cas de SNC multiples. Nous considérons dans la suite du papier, que la priorité va à c_{dec} puis à c_{act}.

3.1 Incompétence de partenariat

Un des buts à atteindre pour un BA est de trouver un partenaire. Si un BA ba_i rencontre, dans une cellule, un autre BA ba_j avec lequel il ne peut signer de partenariat, ba_i est *incompétent*, en utilisant la terminologie AMAS (Capera *et al.*, 2003). Par exemple, un BA représentant un cours d'un enseignant rencontre un autre BA représentant un cours d'un autre enseignant. Cette NCS d'incompétence de partenariat doit conduire vers une réorganisation collective. Comme la seule entité capable de détecter cette situation est l'agent lui-même, ce dernier est le seul à pouvoir changer l'état de l'organisation. Dans ce cas, l'agent a besoin de changer sa position pour rencontrer d'autres partenaires potentiels. De plus, pour permettre une exploration plus efficace des possibilités de partenariat, ba_i mémorise l'adresse et les BAs connus de ba_j pour les échanger au cours de futures rencontres. Cette règle d'auto-organisation coopérative s'exprime comme suit :

Nom : **Incompétence de partenariat** (pour ba_i)

Condition : $\exists j(j \neq i \wedge knows(ba_i, ba_j) \wedge (\neg compatible(ba_i, ba_j) \vee (pCost(ba_i, ba_j) \geq pCost(ba_i, partnership(ba_i)))))$

Actions : `memorize(ba`$_i$`,knows(ba`$_j$`)); move`

La comparaison des coûts de partenariats ($pCost$) permet à ba_i de décider si le nouveau partenariat potentiel avec ba_j est moins contraignant que le partenariat courant.

3.2 Incompétence de réservation

Similairement à l'incompétence de partenariat, les BAs doivent être capables de changer l'organisation lorsque leur réservation n'est pas pertinente. Cette NCS d'incompétence arrive quand un BA ba_i occupe une cellule dont les contraintes ne sont pas compatibles avec les siennes. Par exemple, un BA représentant un cours d'un enseignant est dans une cellule ($cell(ba_i)$) représentant une salle n'ayant pas assez de places pour accueillir le cours. Dans ce cas, ba_i doit se déplacer dans la grille pour explorer l'espace des possibilités de réservations :

Nom : **Incompétence de réservation** (pour ba_i)

Condition :
$\neg compatible(ba_i, cell(ba_i)) \vee (rCost(ba_i, cell(ba_i)) \geq rCost(ba_i, reservation(ba_i)))$

Actions : `memorize(ba`$_i$`,cell(ba`$_i$`)); move`

Pour améliorer l'exploration de la grille, les BA mémorisent les cellules qu'ils ont parcourues dans lesquelles une NCS est survenue pour les partager lors de rencontres futures et afin de les éviter lors de leur exploration à venir.

3.3 Conflit de partenariat

Les situations dans lesquelles un BA désire signer un partenariat avec un autre BA qui a déjà un partenaire peuvent arriver. Ces situations sont des conflits de partenariat qui violent la condition c_{act} des AMAS. Par conséquent les agents doivent être capable de les détecter et de les résoudre :

Nom : **Conflit de partenariat** (pour ba_i)

Condition :
$\exists j \exists k (j \neq i \wedge i \neq k \wedge knows(ba_i, ba_j) \wedge compatible(ba_i, ba_j) \wedge partnership(ba_j) = ba_k)$

Actions :

```
if (pCost(ba_i,ba_j) < pCost(ba_i,partnership(ba_i)))
   then partner(ba_i,ba_j)
   else move
```

Dans ce cas, la coopération est directement incluse dans l'action de résolution : le partenariat sera signé avec l'agent qui a le plus de difficultés à trouver des partenaires (en comparant les $pCost$).

Lorsqu'il signe un partenariat, un BA informe son partenaire précédent et son proxy. Comme cet algorithme est distribué, l'action `partner(ba`$_i$`,ba`$_j$`)` doit être atomique (au sens de l'accès à une section critique), mais est composé des instructions suivantes :

```
unpartner(ba_i,partnership(ba_i));
setPartner(ba_i,ba_j);
inform(partnership(ba_i));
inform(proxy(ba_i))
```

3.4 Conflit de réservation

Comme pour le partenariat, la réservation peut conduire à un conflit : un BA désire réserver une cellule déjà réservée. Un conflit de réservation peut être spécifié comme suit :

Nom : **Conflit de réservation** (pour ba_i)

Condition :
$\exists j(j \neq i \wedge (reservation(ba_j, cell(ba_i)) \vee \exists k(reservation(ba_j, c_k) \wedge slot(cell(ba_i)) = slot(c_k) \wedge proxy(ba_i) = proxy(ba_j))) \wedge compatible(ba_i, cell(ba_i))$

Actions :

```
if (rCost(ba_i,cell(ba_i)) < rCost(ba_i,reservation(ba_i)))
   then book(ba_i,cell(ba_i))
   else move
```

Lorsqu'il réserve une cellule, un BA doit avertir son partenaire et son proxy afin de les informer de ne pas réserver au même créneau horaire. Comme pour l'action `partner(ba_i,ba_j)`, l'action atomique `book(ba_i,cell(ba_i))` est composée des instructions suivantes :

```
unbook(ba_i,reservation(ba_i));
setBook(ba_i,cell(ba_i));
inform(partnership(ba_i));
inform(proxy(ba_i))
```

3.5 Inutilité de réservation

Dans le cas où un BA est dans la même cellule que l'un de ses frères (même proxy), réserver est inutile. Par conséquent, il peut quitter la cellule sans l'analyser :

Nom : **Inutilité de réservation** (pour ba_i)

Condition : $cell(ba_i) = cell(partnership(ba_i))$

Actions : `processEncounteredBAs(); move`

Analyser les BAs rencontrés (`processEncounteredBAs()`) correspond à l'analyse de la mémoire (liste) des BAs précédemment rencontrés ou partagés par d'autres agents.

4 Prototypage et expérimentations

Pour valider l'algorithme distribué que nous proposons, un prototype d'ETTO a été développé et plusieurs tests ont été effectués pour souligner l'influence de la cardinalité (le nombre d'agents), le bénéfice de la résolution dynamique et la robustesse. Les expérimentations sont basées sur un cahier des charges pour l'emploi du temps établi par le groupe de travail ASA de l'AFIA[2]. Ce cahier des charges est décomposé en quatre variantes : du simple problème sans relâchement de contrainte aux systèmes ouverts par ajout et suppression d'agents en cours de résolution. Pour chacune d'entre-elles, nous obtenons une solution – non unique dans bien des cas.

[2] `http ://www-poleia.lip6.fr/~guessoum/asa/BenchEmploi.pdf`

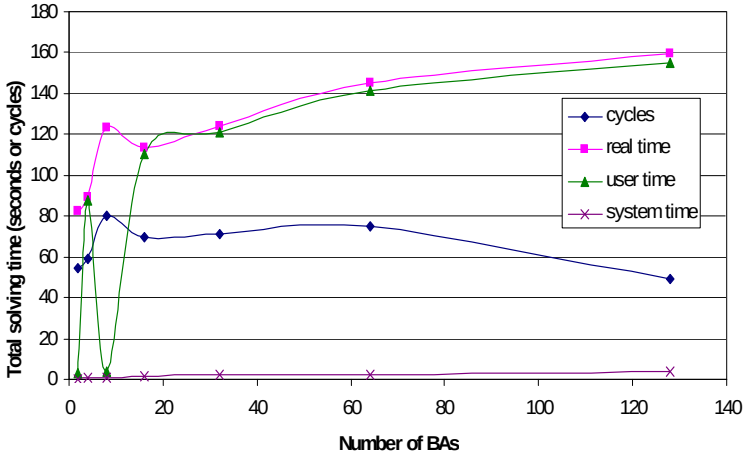


FIG. 1 – Variation du temps de résolution en fonction du nombre d'agents BAs.

4.1 Influence de la cardinalité

La coopération est un liant collectif pour améliorer les interactions entre agents. Ainsi, elle devient un critère pertinent de réorganisation dans des systèmes ayant une cardinalité minimum. La figure 1 montre l'évolution du temps de résolution en fonction du nombre d'agents dans le système. Pour ces expérimentations, nous gardons la même taille d'espace de recherche en augmentant le nombre de cellules de la grille en fonction du nombre d'agents. Seules des contraintes de disponibilités sont attachées aux enseignants : un créneau horaire par jour est interdit.

Une fois le maximum atteint (à 8 BAs), le nombre de cycles (durant lesquels chaque agent agit une fois) décroît lorsque le nombre d'agents augmente. Comme l'espace garde la même dimension, les agents trouvent facilement des partenaires. La mesure de temps qui varie le moins est le temps réel d'exécution. Par conséquent, c'est l'indicateur le plus pertinent de l'évolution du temps de résolution. Au-delà de 32 agents, il suit une courbe logarithmique. Plus il y a de BAs dans le système, plus la résolution est efficace – si une solution existe.

4.2 Relâchement de contraintes

Dans ces expérimentations, les agents doivent relâcher des contraintes pour trouver une solution. La figure 2 montre l'efficacité de la résolution d'ETTO pour une variante à 36 agents (BAs) nécessitant un relâchement de contraintes. Les réservations sont établies après les partenariats. ETTO trouve une solution avec un coût de contraintes relâchées de 10 en 265 cycles. Ce coût représente la somme des poids de toutes les contraintes relâchées par les agents. Néanmoins, le prototype actuel ne gère pas le partage coopératif de cellules lors des négociations et par conséquent lorsqu'un BA se déplace il le fait de manière aléatoire.

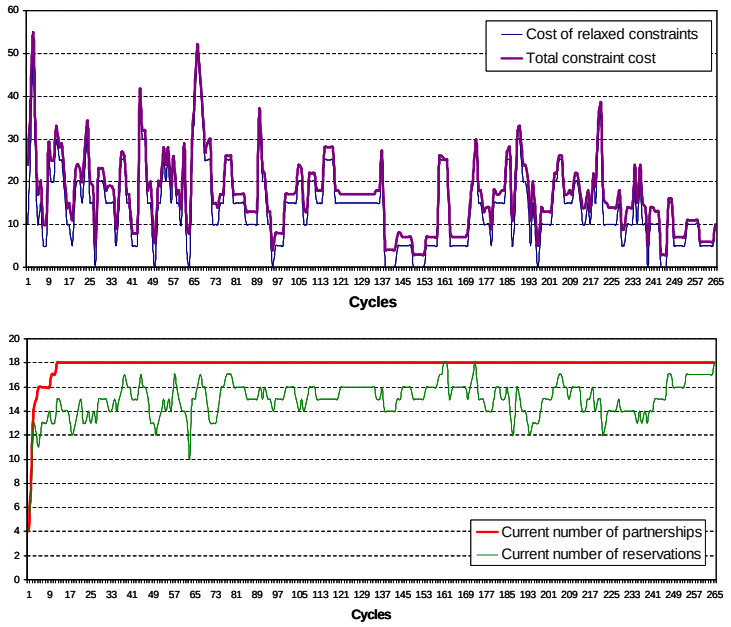


FIG. 2 – Variation du coût global (*en haut*) et des partenariats (*en bas*) durant la résolution d'un problème nécessitant une relaxation de contraintes.

4.3 Résolution dynamique

Les deux premières expérimentations montrent l'apport de l'utilisation de la coopération pour obtenir une résolution efficace de l'emploi du temps. Nous avons effectué une troisième expérimentation pour tester la robustesse à la dynamique de l'environnement. Ici les contraintes apparaissent et disparaissent en cours de fonctionnement. Les contraintes de disponibilités des acteurs ou des salles peuvent aussi évoluer. De plus, des agents peuvent apparaître ou disparaître. En prenant en compte la modélisation choisie, ajouter des contraintes n'est pas différent d'ajouter des agents qui portent des contraintes.

La figure 3 montre des résultats d'une expérimentation avec 36 BAs initiaux. Au cycle 364, à la stabilisation du système, 8 BAs sont retirés de la grille, augmentant ainsi le coût des contraintes relâchées. 20 cycles plus tard, 8 nouveaux agents sont placés aléatoirement dans la grille avec des contraintes adéquates. Le système effectue seulement 7 cycles (du 384 au 391) pour retrouver une organisation adéquate avec un coût nul.

4.4 Discussion

Les problèmes de résolution d'emplois du temps universitaires sont de véritables problèmes dynamiques. Recommencer la résolution de zéro à chaque modification de

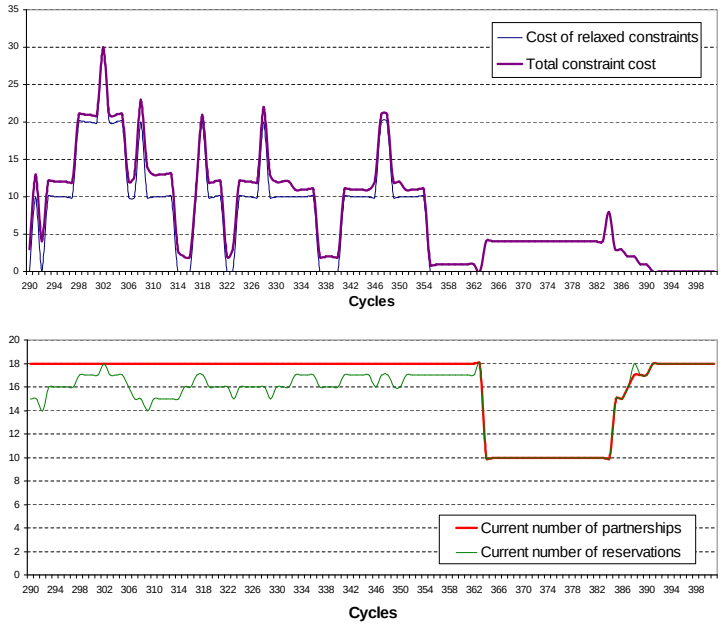


FIG. 3 – Variation du coût global (*en haut*) et des partenariats (*en bas*) durant la résolution d'un problème nécessitant une relaxation de contraintes avec suppression d'agent après la stabilisation du système.

contrainte peut être inefficace, voire infaisable en temps réel. Habituellement, l'objectif principal est d'obtenir un impact minimal sur la solution courante comme dans (Müller & Rudova, 2004) où le problème est résolu en introduisant un nouvel algorithme qui limite le nombre de perturbations additionnelles. Cambazard et al gèrent des problèmes dynamiques via une programmation par contraintes avec explications, plus particulièrement, de nouveaux opérateurs sont donnés pour effectuer une nouvelle propagation lorsqu'une contrainte est enlevée et que ses effets passés sont défaits (Cambazard *et al.*, 2004). Dans ETTO, dès qu'une contrainte est ajoutée ou supprimée pour un agent, ce dernier questionne ses réservations et son partenaire ; s'il juge qu'elles sont inconsistantes avec le nouvel état, il essaie de trouver une autre réservation ou un autre partenaire en explorant la grille et en suivant son comportement de base. L'intérêt est que le coût de la solution et le plus équitablement réparti possible. Si un nouvel agent est ajouté, il commence immédiatement à chercher un partenaire et une réservation, et s'il est supprimé alors toutes ses contraintes et réservations sont supprimées, ses frères et son partenaires sont prévenus. Le principal atout d'ETTO est que les modifications peuvent être faites sans arrêter la recherche pour une solution en progression. Les événements provenant des acteurs sont pris en charge à la volée. De plus, cette capacité à introduire des agents nous a permis de montrer que l'ajout d'agent surnuméraire apporte en efficacité.

Mais ETTO possède des points faibles. Par exemple, résoudre des problèmes surcontraints n'est pas totalement efficace car si les agents ont trouvé une solution, ils continueront à explorer la grille pour trouver une solution plus pertinente. Comme les agents n'ont qu'une perception limitée de leur environnement, ils ne peuvent prendre en compte le coût global des contraintes pour arrêter leur recherche. Pour utiliser ETTO, nous considérons qu'il existe un oracle (l'humain responsable des emplois du temps) qui arrêtera le processus de résolution lorsque l'organisation répondra à ses besoins – avec un coût minimal de contrainte, par exemple. De plus, la recherche pour une cellule dans la grille n'est pas efficace car effectuée de manière aléatoire. Pour le moment, nous ne nous intéressons pas à l'efficacité, nous voulons juste montrer que notre approche par auto-organisation coopérative peut produire des résultats positifs. Néanmoins, la prochaine étape de notre travail est d'améliorer ETTO en fournissant la gestion coopérative de la mémoire des cellules afin d'explorer la grille plus efficacement.

Enfin, nous avons choisi un exemple simple pour appliquer ETTO. Une de nos perspectives est de se baser sur un problème plus complexe ou sur un cahier des charges comme celui donné par le réseau *Metaheuritics Network*[3]. Ceci nous permettra de comparer notre approche à d'autres, conceptuellement proches, comme les algorithmes génétiques, le recuit simulé (mais qui reste à un point de vue global sur le système et sa fonction) ou les algorithmes fourmis, intrinsèquement plus distribués.

5 Conclusion

Nous pensons que le caractère distribué inhérent au problème d'emplois du temps universitaires justifie l'approche par multi-agent pour le résoudre. Contrairement à d'autres travaux, avoir un protocole de négociation entre agent n'est pas la seule possibilité pour trouver une solution. Nous avons proposé une solution basée sur les systèmes multi-agents adaptatifs dans lesquels la coopération est un critère pour le changement des interactions entre agents afin de faire émerger la fonction globale du système. Ce type de programmation peut être assez efficace pour résoudre des problème complexes, peu ou mal spécifiés, et pour lesquels les concepteurs ne possèdent pas d'algorithme préétabli. Ceci est montré par les résultats préliminaires obtenus grâce au prototype d'ETTO aussi bien que par des travaux antérieurs sur la résolution de problèmes divers.

Remerciements

Nous tenons à remercier les acteurs d'ETTO et de son insertion dans ADELFE[4] : Carole Bernon, Valérie Camps Marie-Pierre Gleizes, François Bérenger et Sylvain Peyruqueou.

Merci aussi aux relecteurs pour leurs critiques avisées sur le fond et la forme qui nous ont permis d'améliorer la présentation de notre travail.

[3] `http ://www.metaheuristics.org`

[4] `http ://www.irit.fr/ADELFE`

Références

ALI S., ZIMMER R. & ELSTOB C. (1997). The question concerning emergence : Implication for Artificiality. In DUBOIS, D.M., Ed., *1^{st} CASYS'97 Conference*.

BISTARELLI S., FARGIER H., MANTANARI U., ROSSI F., SCHIEX T. & VERFAILLIE G. (1999). Semiring-based constraints CSPs and valued CSPs : frameworks, properties, and comparison. *Constraints : an International Journal*, **4**(3), 199–240.

BONABEAU E., THERAULAZ G., DENEUBOURG J.-L., ARON S. & CAMAZINE S. (1997). Self-organization in social insects. *Trends in Ecology and Evolution*, **12**, 188 –193.

CAMBAZARD H., DEMAZEAU F., JUSSIEN N. & DAVID P. (2004). Interactively Solving School Timetabling Problems using Extensions of Constraint Programming. In *Proc. of the 5th International Conference of the Practice and Theory of Automated Timetabling (PATAT), Pittsburg, USA*.

CAMPS V., GLEIZES M.-P. & GLIZE P. (1999). A Theory of Emergent Computation Based on Cooperative Self-Organization for Adaptive Artificial Systems. In *4^{th} European Congress of Systems Science, Valencia*.

CAPERA D., GEORGÉ J., GLEIZES M.-P. & GLIZE P. (2003). The AMAS theory for complex problem solving based on self-organizing cooperative agents. In *1^{st} International TAPOCS Workshop at IEEE 12th WETICE*, p. 383 –388 : IEEE.

DECHTER, MEIRI & PEARL (1991). Temporal constraint networks. *Artificial Intelligence*, **49**, 61–95.

GEORGÉ J.-P., EDMONDS B. & GLIZE P. (2004). Making Self-Organising Adaptive Multiagent Systems Work. In F. BERGENTI, M.-P. GLEIZES & F. ZAMBONELLI, Eds., *Methodologies and Software Engineering for Agent Systems (Chapter 16)*, p. 321 –340 : Kluwer.

GOLDSTEIN J. (1999). Emergence as a Construct : History and Issues. *Journal of Complexity Issues in Organizations and Management*, **1**(1).

KOHONEN T. (2001). *Self-Organising Maps*. Springer-Verlag.

MÜLLER T. & RUDOVA H. (2004). Minimal Perturbation Problem in Course Timetabling. In *Proc. of the 5th International Conference of the Practice and Theory of Automated Timetabling (PATAT), Pittsburg, USA*.

SOCHA K., KNOWLES J. & SAMPELS M. (2002). A MAX-MIN Ant System for the University Timetabling Problem. In *Proceedings of 3^{rd} International Workshop on Ant Algorithms, ANTS'02*, volume 2463 of *LNCS*, p. 1 –13 : Springer-Verlag.

YOKOO M., DURFEE E., ISHIDA Y. & KUBAWARA K. (1998). The Distributed Constraint Satisfaction Problem : Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, **10**, 673–685.

Contrôle Dynamique d'Agents Autonomes

Caroline Chopinaud[1],[2]

[1] Thales Systèmes Aéroportés - 1 boulevard Jean Moulin 78852 Elancourt

[2] Laboratoire Informatique de Paris 6 - 8 rue du Capitaine Scott 75015 Paris
caroline.chopinaud@fr.thalesgroup.com

Résumé : Pouvoir avoir confiance dans le comportement d'un système est primordial, en particulier dans le contexte d'applications critiques telles que les systèmes embarqués ou les systèmes temps réels. Notre objectif est d'assurer qu'un système multiagent a un comportement en accord avec ce que l'on peut attendre de lui. L'utilisation de méthodes statiques pour la validation ne permet pas d'identifier toutes les situations où le système, une fois mis en condition réelle, sera susceptible de générer des erreurs. Nous proposons une approche complémentaire d'auto-surveillance et d'auto-régulation permettant aux agents de vérifier la cohérence de leur comportement au cours de leur exécution et de réagir en conséquence. Notre approche consiste à fournir aux agents un ensemble de lois qu'ils devront respecter. Ce papier présente un "framework" qui, à partir d'un modèle d'agent, d'une description de comportement et de lois, va créer des agents capables de s'auto-contrôler. Pour ce faire, le générateur modifie le programme des agents en insérant des points de contrôle qui vont produire des événements relatifs au comportement de l'agent. Le générateur fournit aux agents une architecture particulière leur permettant de surveiller leur comportement à l'aide des événements produits. Les principes de ce générateur sont illustrés sur un exemple.

Mots-clés : Systèmes Multiagents, Contrôle, Vérification, Autonomie.

1 Introduction

L'autonomie est une caractéristique essentielle des agents cognitifs. Nous la verrons comme la capacité d'un agent à prendre seul ses décisions, sans qu'une entité extérieure prenne part à son processus de choix (Barber & Martin, 1999). Du point de vue du programmeur cela signifie que la mise en place d'un agent nécessite de prendre en compte le fait que le comportement des autres agents ne peut être prédit avec certitude. Il faut donc programmer son agent en ayant à l'esprit que les autres peuvent ne pas réagir de la façon attendue. Aussi, une telle vision pose-t-elle le problème de la confiance que l'on peut avoir dans le fonctionnement de son système. Dans les milieux industriels concernés par des applications critiques, telles que les systèmes embarqués ou les systèmes temps réels, la mise en oeuvre de tels systèmes risque de soulever des objections en raison de leur imprévisibilité. Il est donc indispensable de pouvoir garantir qu'un SMA et les agents le constituant respectent des exigences de comportement

préétablies, importantes pour le bon fonctionnement de l'application, tout en préservant au mieux leur autonomie.

L'objectif de nos travaux est d'assurer qu'un SMA a un comportement en accord avec ces exigences. Une première approche serait d'utiliser des méthodes classiques de validation des systèmes que l'on appliqueraient aux systèmes multiagents, telles que les techniques de tests, de model-checking et de démonstration automatique. Mais, outre le fait que ces techniques ne sont jamais en mesure de détecter toutes les erreurs possibles d'un système (Schnoebelen, 1999), elles ne sont pas forcément adaptées à la validation des systèmes multiagents dans leur globalité. Bien que ces techniques soient indispensables à la validation d'un SMA, elles ne sont pas suffisantes car elles laissent la possibilité d'apparition d'erreurs une fois le système mis en condition réelle d'exécution, du fait de leur application sur un modèle abstrait du système et de son environnement. Pour détecter ces erreurs il est intéressant d'effectuer une vérification dynamique du comportement du système en complément des méthodes classiques. Cette vérification[1] consiste en la surveillance et la régulation des comportements incohérents dans le but d'empêcher que le système échoue.

De plus nous pensons que les agents sont les mieux placés pour effectuer le contrôle de leur comportement. Nous fournissons aux agents les moyens nécessaires à la surveillance de leur comportement. Grâce à leur capacité de raisonnement, ils vont pouvoir se réguler pour revenir à un état cohérent. Le surveillance et la régulation de comportement seront effectuées par les agents eux-mêmes.

Bien qu'il soit possible pour un programmeur d'inclure le code de contrôle nécessaire à la surveillance et à la détection d'erreur au sein même des agents, cela peut s'avérer complexe. L'instrumentation manuelle du code d'un système, pour y insérer des détecteurs d'évènements particuliers, est difficile et enclin à l'erreur (Mansouri-Samani, 1995). De plus, lorsqu'une propriété à valider concerne plusieurs agents, il apparaît comme impossible de trouver la cause et de résoudre le problème au moyen d'une implémentation simple et claire. Aussi devient-il difficile de faire évoluer le comportement des agents mais également le contrôle auquel ils sont soumis. En se basant sur les instrumentations automatiques pour le monitoring de systèmes distribués, il est envisageable d'automatiser l'instrumentation des programmes de comportement des agents pour mettre en place le contrôle, et ainsi faciliter le travail du programmeur.

Nous proposons un générateur qui, à partir d'une description des exigences associées aux agents et du code de leur comportement, va créer des agents capables d'autocontrôle. Dans une première partie nous allons présenter les principes de notre approche, pour ensuite nous intéresser à l'architecture de notre générateur. Nous illustrerons principes et fonctionnements sur un exemple.

2 Vérification de systèmes

Notre objectif étant de garantir qu'un système aura un comportement en accord avec ce que l'on peut attendre de lui, il est intéressant d'introduire avant tout quelques-unes

[1] Nous emploierons par la suite le terme de contrôle pour désigner cette vérification.

des techniques de vérification de logiciel afin de mettre en avant ce qui semble se rapprocher le plus de ce que nous souhaitons appliquer aux agents.

2.1 Techniques de Validation

Il existe différentes techniques pour valider un système, comme les tests, le model checking et la démonstration automatique. Ces différentes approches sont indispensables mais elles laissent subsister des situations dans lesquelles peuvent apparaître des erreurs une fois le système mis en condition réelle d'exécution. Les tests ne peuvent être exhaustifs, le model checking, travaillant sur une abstraction du système, ne peut que détecter les erreurs de ce modèle et la détection automatique est lourde et complexe. Néanmoins ces techniques ont fait leurs preuves dans la validation des systèmes multiagents. B.Edmons et J.J.Bryson (Edmonds & Bryson, 2004) mettent en avant l'utilisation des tests au détriment des méthodes formelles, en raison des problèmes liés à l'abstraction et la modélisation nécessaires pour les appliquer. M.Wooldridge (Wooldridge *et al.*, 2002) et M.Benerecetti (Benerecetti *et al.*, 1997) montrent les possibilités de validation des SMA grâce au model checking, mais pour des situations particulières (agents construits suivant un modèle particulier comme BDI (Belief, Desir, Intention) par exemple ou validation de protocoles). Du fait des difficultés à valider un comportement distribué et non déterministe (Joyce *et al.*, 1987) et des risques d'explosion du nombre d'états de sa modélisation, il n'est de toute façon pas envisageable de valider un SMA dans sa globalité,. Enfin certains se sont penchés sur la démonstration automatique, mais bien qu'intéressante, cette approche ne peut être généralisable à tout type d'agents et de problèmes (Burkhard, 1993).

Nous ne pouvons donc nier l'intérêt de ces techniques, mais nous considérons que pour avoir une totale confiance dans l'exécution d'un SMA il est indispensable de s'attacher à une vérification en-ligne du système. De ce fait nous nous plaçons dans le cadre suivant : un développeur a suivi l'ensemble des spécifications du système pour mettre en place les agents et a tout mis en oeuvre pour valider un SMA. Mais pour garantir que le comportement de ce dernier suivra les attentes des clients et utilisateurs, il est nécessaire de faire une vérification du comportement du système en cours d'exécution. Pour cela, il est intéressant d'utiliser les techniques de monitoring appliquées aux systèmes distribués.

2.2 Monitoring de systèmes distribués

Le monitoring est une technique pour observer et comprendre le comportement dynamique de programmes à l'exécution. Il existe trois types de monitoring, le monitoring hardware, software et hybride. Dans le monitoring hardware ce sont des objets séparés qui sont utilisés pour détecter les événements associés à un objet ou un groupe d'objets. Ces monitors hardware effectuent leur détection par l'observation des bus du système ou en utilisant des sondes physiques connectées aux processeurs ou aux canaux d'E/S. Un monitor software partage les ressources avec le système sous surveillance. Le programme est instrumenté en insérant des sondes logicielles dans le code pour détecter les événements. Enfin, le monitoring hybride consiste en un dispositif hardware qui reçoit

les informations de surveillance générées par les sondes logicielles insérées dans le système surveillé (de Sousa Dias & Richardson, 2002). Nous nous intéressons plus particulièrement au monitoring software. Pour surveiller le comportement d'un système il faut donc insérer dans le programme des détecteurs d'évènements. L'instrumentation peut être faite manuellement par le programmeur (Marzillo *et al.*, 1991) ou de façon automatique. L'automatisation de l'insertion peut prendre deux formes, soit le programmeur va utiliser un méta-langage (Lumpp *et al.*, 1990) ou une librairie de routines (Huang & Kintala, 1995) permettant d'insérer les détecteurs de façon transparente, soit l'insertion des détecteurs se fera à la compilation à partir de spécifications des événements (Liao & Cohen, 1992). C'est cette dernière vision qui a attirée notre attention et que nous souhaitons appliquer aux agents dans la perspective de réduire réduire le travail du programmeur au maximum, en ce qui concerne la prise en charge du contrôle.

3 Contrôle d'agents autonomes

Contrôler un SMA revient à surveiller son comportement en cours d'exécution et à le réguler en cas de détection d'anomalies, les agents surveillant et régulant leur propre comportement. L'idée est donc d'insérer automatiquement un certain "code de contrôle" à l'intérieur des agents pour leur permettre de vérifier la concordance de leur comportement avec les exigences associées.[2]

3.1 Vérification de comportement

Nous avons vu la difficulté de modéliser le comportement d'un SMA et des agents le constituant. Les agents étant construits à partir d'un ensemble d'exigences, en nous basant sur des travaux de M.S.Feather (Feather *et al.*, 1998), nous pensons qu'elles peuvent être directement ou indirectement utilisées pour représenter ce qui est correct ou non dans le comportement et l'état interne d'un agent. Ces exigences sont, ce qu'on appellera par la suite, les **lois** du système. Ainsi un agent capable d'auto-contrôle, s'assure que les lois qui lui sont attribuées sont respectées tout au long de son exécution. Mais cette surveillance ne suffit pas, lorsque l'agent détecte qu'une loi n'est pas satisfaite, il se doit de réguler son comportement en conséquence, à partir d'informations de réparation.

3.2 Aspects principaux

Avant tout nous souhaitons simplifier le travail du programmeur. Il ne va donc pas se charger d'instrumenter le code des agents pour savoir s'ils respectent bien l'ensemble des lois. Nous fournissons un générateur, qui à partir du code du comportement des agents et de l'ensemble des lois va créer des agents en mesure d'effectuer leur propre contrôle, en instrumentant automatiquement leur code de comportement.

[2] Ce code de contrôle étant préalablement validé, il n'existe pas de risque que les erreurs viennent de cette partie de l'agent.

Nous souhaitons également que la personne chargée de définir et décrire les lois ne soit pas obligatoirement le programmeur. Le client, ou l'utilisateur, doit aussi être en mesure de poser des lois sur le système, car c'est lui qui a défini les exigences et qui va exécuter le système dans des conditions réelles. Or un client ou un utilisateur ne peut réellement connaître ou comprendre l'implémentation des agents. Pour qu'ils puissent poser des lois sur un système, celles-ci doivent être décrites à un niveau tel que les choix d'implémentation n'aient pas à rentrer en considération.

Enfin, nous ne voulons pas nous restreindre à un contrôle sur un seul type de modèle d'agent. Il ne serait pas réaliste de n'en considérer qu'un seul, nous avons pour ambition de permettre le contrôle d'un grand nombre de SMA. De plus, les agents d'un même système peuvent être construits suivant des modèles différents et, dans ce cas, une loi doit pouvoir être appliquée à ces agents sans avoir à l'adapter en conséquence. Il faut se placer à un niveau d'abstraction permettant d'englober un grand nombre de modèles d'agent.

Les lois doivent donc porter sur des concepts généraux, représentatifs des différents modèles utilisés mais aussi de l'application. Les modèles utilisés pour concevoir des agents sous contrôle doivent, dès lors, fournir un descriptif des concepts qui font leurs spécificités. C'est à partir de cet ensemble de concepts qu'il sera possible de construire les lois.

3.3 Mise en place du contrôle

Nous pouvons diviser le mise en place du contrôle en quatre étapes :

(1) Le développeur du modèle doit fournir une description des concepts le caractérisant ainsi que leur correspondance dans le programme pour permettre la surveillance de leur état au cours de l'exécution.

(2) Le client (ou l'utilisateur) doit fournir un ensemble de lois qu'il souhaite voir être vérifiées tout au long de l'exécution du système. Ces lois peuvent concerner un ou plusieurs agents. Elles portent sur les concepts décrits à l'étape 1.

(3) Le développeur du système construit les agents en suivant les exigences préétablies.

(4) Le générateur traite les lois, récupère les liens aux concepts pour injecter le code de contrôle et génère les agents. Ces derniers s'auto-contrôlent pour détecter s'ils respectent les lois qui leur sont attribuées.

4 Un Exemple

Dans le but d'illustrer notre approche et le fonctionnement de notre générateur nous allons introduire un exemple.

4.1 Le système multiagent

Le système multiagent est constitué au départ de trois agents : un agent A, se chargeant de discuter avec un utilisateur ; un agent B, se chargeant de gérer les problèmes

proposés par l'agent A ; un agent C, se chargeant de résoudre les problèmes. Lorsque l'agent B reçoit un problème il l'analyse et demande à l'agent C s'il peut effectuer la résolution. Si l'agent C est disponible pour cette tâche, l'agent B lui transmet le problème, dans le cas contraire l'agent B va créer un nouvel agent de type C et lui transmettre le problème. Une fois le problème résolu, l'agent C renvoie le résultat à l'agent B qui va le transmettre à l'agent A pour le fournir, dans un format compréhensible, à l'utilisateur.

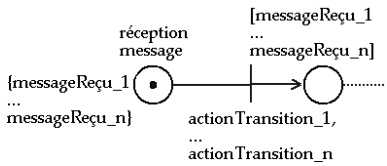


FIG. 1 – Réseau de Petri modélisant les agents

Chaque agent est construit sur un modèle de réseau de Petri (figure 1). A chaque état, les agents se mettent en attente de *réception d'un nouveau message*. Suivant le message reçu, le réseau va déclencher une *action particulière* et se mettre dans un nouvel état. Les actions observables sont, par exemple : la réception de message (au niveau des états), les actions de transition (l'envoi d'un message, la création d'un nouvel agent, l'affichage d'une solution, le traitement d'un problème). Les données observables sont, par exemple : l'état d'un agent, les messages reçus.

4.2 Les lois du système

L'utilisateur souhaite que certains comportements ou états de données au sein du système soient bien respectés. Par exemple, il peut être souhaitable que l'agent A n'envoie pas des demandes de résolution de problème à l'agent B à une cadence trop élevée. Dans le cas contraire, l'agent B pourrait ne plus arriver à suivre les demandes et par conséquent les agents C pourraient ne plus répondre en un temps acceptable aux problèmes proposés. Dans le cas où cette exigence n'est pas respectée l'utilisateur précise que l'envoi du message incriminé doit être annulé. L'utilisateur peut décider de poser la loi suivante :

L1 : "Un agent de type A ne doit pas envoyer des messages, à un agent de type B, à une cadence supérieur à 1 Hz."

Un autre problème pouvant perturber le fonctionnement du système peut apparaître au niveau de la création d'agent C par l'agent B. Ce dernier, demande aux agents C qui est disponible. Si au bout d'un temps *t* donné, l'agent B ne reçoit aucune réponse, il en déduit qu'aucun agent est disponible et en créé un nouveau. Si le temps d'attente n'est plus adapté, l'agent B va se mettre à créer des agents C quasiment à chaque soumission d'un problème. Le système peut alors se trouver en surcharge et empêcher la bonne exécution de l'application. Une action de réparation lorsque cette exigence n'est plus

vérifiée pourrait être de bloquer la création du nouvel agent. Une loi pour détecter ce problème serait :

L2 : "Un agent de type B ne doit pas lancer la création d'un agent de type C s'il en existe un de ce même type qui est dans un état disponible ".

Enfin le client veut pouvoir vérifier que certains protocoles qu'il a définis pour le système sont effectivement suivis. Par exemple l'agent C peut s'attendre à recevoir les messages suivants, dans cet ordre : l'identifiant du problème, le temps maximum d'attente du résultat, les données d'un problème. Si ce protocole n'est pas suivi par l'expéditeur, il se peut que l'agent C ne puisse pas résoudre le problème ou échoue. Dans le cas où le protocole n'est pas respecté une solution est de prévenir l'agent expéditeur de l'erreur. Pour détecter l'apparition de ce problème une loi serait :

L3 : Un agent C doit recevoir le message de durée puis le message d'identification puis les données du problème.

5 SCAAR : un framework pour la génération d'agents autonomes auto-contrôlés

SCAAR (Self-Controlled Autonomous Agent geneRator) est un "framework" permettant la génération d'agents capables de s'auto-contrôler. Pour cela, il utilise l'ensemble des lois associées aux différents agents d'un SMA, l'ensemble des concepts utilisés dans ces lois, et injecte dans les programmes des agents, le code de contrôle nécessaire à l'analyse du bon respect des lois. La figure 2 représente l'architecture de notre "framework". Sa composition est la suivante :

- **Ontologie :** L'ensemble des concepts représentant le modèle utilisé pour décrire l'agent et ceux caractéristiques du SMA.
- **Lois :** L'ensemble des propriétés que l'agent doit respecter.
- **Modèle d'agent (Lien) :** Les liens entre la description du modèle (ensemble des concepts) et l'implémentation du modèle.
- **Générateur :** Le créateur de l'agent final à partir des éléments précédents.
- **Agent Auto-Contrôlé :** L'agent final exécutable. Il surveille son propre comportement afin de vérifier s'il respecte les lois et le régule si une loi est transgressée.

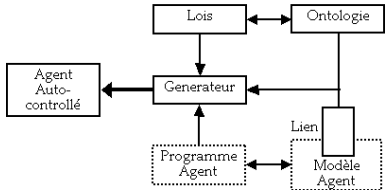


FIG. 2 – Architecture du "framework" SCAAR

5.1 L'ontologie

Nous avons vu que les lois devaient porter sur des concepts de haut niveau permettant de décrire les spécificités des agents et du système. Nous avons donc cherché à mettre en place une ontologie de base représentative des modèles connus pouvant être utilisés pour construire des agents. Un ensemble de concepts d'agent est proposé par D.N.Lam et K.S.Barber (Lam & Barber, 2004) dans le cadre de la vérification d'agent. Elle contient les concepts de *But*, *Croyance*, *Intention*, *Action*, *Événement*, *Message*. Nous reprenons une partie de ces concepts (*But*, *Action*, *Message*) et nous ajoutons d'autres concepts qui nous semble plus caractéristiques des agents vis-à-vis de l'étude que nous avons faites des modèles d'agents (BDI, CLAIM...) (*Objet*, *Connaissance*, *Plan*, *Création d'agent*, *Envoi de message*, *Réception de message*, *Migration*). Enfin comme nous nous plaçons au niveau du SMA, nous ajoutons le concept d'*Agent*.

Chaque concept a un ensemble d'attributs et de méthodes[3] permettant d'exprimer des tests sur les concepts au sein des lois. L'ontologie (Fig. 3) peut être étendue suivant les caractéristiques des modèles utilisés et de l'application.

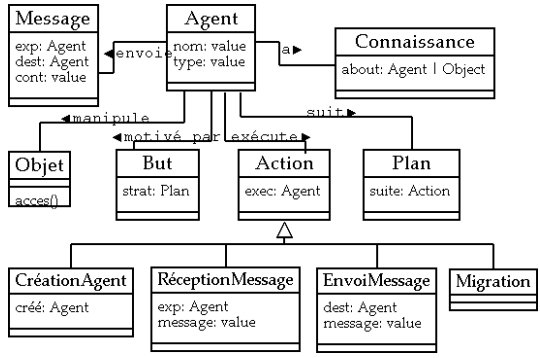


FIG. 3 – Ontologie d'Agent (sous forme de digramme UML)

Les concepts de base que l'on retrouve dans le modèle de notre exemple sont `ReceptionMessage`, `EnvoiMessage`, `CreationAgent`, `Action`, `Agent`. Ceux étendus des concepts de base et représentatifs du modèle sont : `Etat` qui est une `Connaissance` sur soi-même et qui a un paramètre (disponible ou non) et `MessageRecu` qui est un `Message`.

5.2 Les Lois

On peut distinguer deux types de lois :

- Celles décrivant un état ou un comportement non désirés. Elles permettent de détecter des situations où un événement qui ne doit jamais se produire, apparaît.

[3] Elles ne sont pas décrites sur la figure 3

- Celles décrivant un état ou un comportement attendus. Elles permettent de détecter des situations où un événement qui doit se produire, n'a pas lieu.

Nous relions les lois à la logique déontique (von Wright, 1951). Cette dernière a souvent été utilisée pour décrire les normes, qui se rapprochent de notre concept de loi, dans les systèmes multiagents. Ainsi, le premier type de loi correspond à une interdiction et le second à une obligation. Nous nous servons des opérateurs déontiques pour exprimer ce qui est interdit et ce qui est obligatoire et ainsi distinguer leur prise en compte au sein des agents. La structure que nous proposons pour exprimer les lois est néanmoins la même pour les deux types. La loi doit contenir les éléments nécessaires pour savoir quand la déclencher, quand la vérifier, quels concepts sont mis en jeu et quelles propriétés vérifier sur ces concepts. La loi est donc composée :

- **ASSERTION_DEONTIQUE - AD** : Décrit ce qui est obligatoire (O) ou interdit (I). C'est un ensenble d'association entre un agent et un "événement". Cet événement peut être une action à exécuter par un agent (do) ou l'état d'un concept (have). Une association peut avoir des conditions (and).
- **CONDITION_APPLICATION - CAP** : Décrit les conditions à propos du contexte de la loi. Ce peut être une description de quand un événement est interdit ou obligatoire, relatif à un autre événement ou une durée.
- **CONDITION_ALLOCATION - CAL** : Les agents concernés par la loi. Ce sont les agents qui ont une représentation de la loi dans leur comportement, pas seulement les agents qui doivent respecter la loi.
- **REGULATION - REG** : Les actions à exécuter pour sauvegarder le comportement de l'agent quand la loi est transgressée. Ce sont des actions simples portant sur les concepts du système permettant de sauvegarder temporairement les agents concernés (block, execute) et de leur donner la possibilité de réguler leur comportement par la suite.

Si on reprend l'exemple vu à la section 4, nous pouvons écrire les lois suivant cette décomposition :

L1 : "Un agent de type A ne doit pas envoyer des messages, à un agent de type B, à une cadence supérieur à 1 Hz."

(L1) a : Agent, ev1 : EnvoiMessage, ev2 : EnvoiMessage
(AD) **I**(**do**(a,ev2) **and** ev2.dest = ev1.dest)
(CAP) **begin**{ **do**(a,ev1) **and** (ev1.dest).type = typeB) }, **end**{ **begin** + 1 }
(CAL) a.type = typeA,
(REG) **block**{ **do**(a,ev2) }

L2 : "Un agent de type B ne doit pas lancer la création d'un agent de type C s'il en existe un de ce même type qui est dans un état disponible ".

(L2) a1 : Agent, a2 : Agent, ca : CreationAgent
(AD) **I**(**do**(a,ca) **and** (ca.agentCrée).type = typeC)
(CAP) a2 **have** state.value = disponible,
(CAL) a.type = typeB, a2.type = typeC,
(REG) **block**{**do**(a,ca)}

L3 : Un agent C doit recevoir le message de durée puis le message d'identification puis les données du problème.

(L3) a : Agent, mr1 : MessageRecu, mr2 : MessageRecu, mr3 : MessageRecu,
ev : EnvoiMessage
(AD) **O**(**have**(a, mr1.cont = "time(M1)")
then have(a, mr2.cont = "ident(M2)")
then have(a, mr3.cont = "pb(M3)"))
(CAP) null
(CAL) a.type = typeC
(REG) **execute**{ **do**(a, EnvoiMessage(mr1.exp, "pb protocole")}

5.3 Liens au modèle

La génération d'agents auto-contrôlés nécessite l'instrumentation des programmes des agents afin d'insérer les points de contrôle permettant de vérifier que les lois sont respectées. Les concepts sur lesquels portent les lois doivent avoir une signification au niveau du code des agents. Le concepteur du modèle fournit les liens entre les concepts et le code du modèle pour permettre au générateur d'insérer le contrôle à l'intérieur de celui-ci et par extension à l'intérieur des agents.

Voyons sur l'exemple les liens nécessaires pour insérer le code de contrôle au sein des agents. On suppose ici que les agents sont programmés en Prolog. Ils ont accès à un ensemble de primitives permettant d'échanger des messages, de créer des agents et de migrer. Ces primitives sont fournies avec le modèle de réseau de Petri utilisé pour construire les agents.

- ReceptionMessage : dans le modèle cette action correspond à l'appel de la clause de réception de message se situant au niveau des états du réseau.

```
link((receptionMessage == receiveMessage\3),
      receptionMessage.cont == argument(1),
      receptionMessage.dest == argument(2)).
```

- MessageRecu : dans le modèle, ce concept correspond à une variable pour laquelle on fournit un accesseur.

```
link((messageRecu == call(getIncomingMessage\0),
     (messageRecu.cont == call(getContent(messageRecu)),
     (messageRecu.exp == call(getRemote(messageRecu))).
```

5.4 Structure des agents générés

Les agents générés doivent être en mesure de contrôler leur propre comportement. Ils doivent donc avoir une structure appropriée reflétant le caractère introspectif de ce contrôle. Ce dernier consiste, nous l'avons vu, en la surveillance du comportement des agents pour vérifier que des lois sont respectées et la régulation du comportement lorsqu'une loi est transgressée. Les lois pouvant porter sur plusieurs agents, il est important que ceux-ci puissent communiquer entre eux pour vérifier le respect des lois et résoudre les problèmes. Leur structure fournit un moyen de collaboration dans le

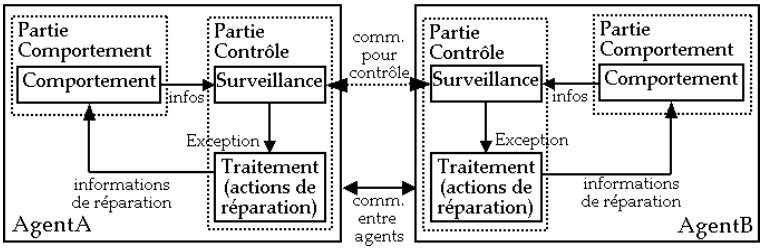


FIG. 4 – Structure des agents générés

contrôle. L'agent généré est divisé en deux parties, une partie correspondant au **comportement** réel de l'agent et l'autre dédiée au **contrôle** et qui va collaborer avec les parties de contrôle des autres agents. La partie de contrôle est divisée à son tour en deux blocs fonctionnels. Un bloc pour la **surveillance** et un pour le **traitement** des lois transgressées. La figure 4 décrit la structure de deux agents auto-contrôlés en interaction. Le comportement envoie des informations à la surveillance sur son exécution et son état interne (i.e. les états des concepts). La surveillance vérifie si les lois sont bien respectées. En cas de transgression d'une loi, un message de détection d'anomalie est envoyé à la partie traitement qui va exécuter les actions et fournir des informations de réparation au comportement. Ce dernier va en déduire les changement à effectuer pour être cohérent.

5.5 Le générateur

Pour créer des agents capables de détecter la transgression des lois au sein de leur propre comportement, le générateur utilise des techniques de tissage pour mettre en place des points de contrôle. Le tissage est une part fondamentale de la programmation par aspect, qui consiste à injecter dans un programme un certain code à un endroit donné. La programmation par aspect utilise le tissage pour injecter des aspects dans les classes d'une application, au niveau des méthodes, de façon à modifier l'exécution finale des classes après compilation. La programmation par aspect, grâce à AspectC++, a été montrée comme particulièrement intéressante pour intégrer du monitoring dans une application (Mahrenholz *et al.*, 2002). Nous reprenons cette idée de tissage pour effectuer une instrumentation automatique des lois et de leur analyse à l'intérieur du programme représentant le comportement d'un agent.

Le générateur va extraire des lois, ce qu'il reconnaît comme étant des concepts faisant partie de l'ontologie associées au SMA. Pour chaque concept il récupère le lien au code de l'agent et va déduire des mots clés du langage de description des lois, le code à insérer au sein de l'agent pour permettre la surveillance. Reprenons par exemple la description de la loi L1 :

L1 : "Un agent de type A ne doit pas envoyer des messages, à un agent de type B, à une cadence supérieur à 1 Hz."

(L1) a : Agent, ev1 : EnvoiMessage, ev2 : EnvoiMessage
(AD) *(1)* **I**(**do**(a,ev2) **and** ev2.dest = ev1.dest)
(CAP) *(2)* **begin**{ **do**(a,ev1) **and** (ev1.dest).type = typeB) },
(3) **end**{ **begin** + 1 }
(CAL) *(4)* a.type = typeA,
(REG) *(5)* **block**{ **do**(a,ev2) }

(1) Tissage avant l'appel à ev2 d'un point de blocage du comportement pour tester si la loi est activée et dans ce cas si elle est respectée.

(2) Tissage d'un point d'activation de la loi juste après l'appel à ev1.

(3) Mise en route d'un compteur à l'activation de la loi pour 1s. Au bout de 1s la loi est désactivée.

(4) Tissage effectuer uniquement dans les agents respectant ces conditions.

(5) Tissage au niveau de ev2 d'un "saut" de l'appel à ev2 à effectuer lorsque la loi est violée.

6 Travaux similaires

Dans les travaux de M.S.Feather (Feather *et al.*, 1998) il est également question de comparer l'exécution d'un SMA à ses spécifications. Dans cette approche, un unique monitor se charge de récupérer les événements envoyés par les agents et un réconciliateur va, en cas de violation, non pas remettre le système dans un état tel qu'il respecterait ses spécifications, mais modifier les spécifications pour qu'elles soient en accord avec le nouveau comportement du système. Ici, il n'est pas question de spécifications primordiales pour le bon fonctionnement du système, les auteurs ne cherchent donc pas à empêcher les mauvais comportements mais à faire en sorte que le système et les spécifications s'adaptent pour rester en accord, tout au long de l'exécution.

D.N.Lam et K.S.Barber (Lam & Barber, 2004) propose une méthodologie, la "Tracing Method"pour tester et expliquer le comportement des agents. L'objectif de cette méthodologie est de s'assurer qu'un agent exécute une action pour de bonnes raisons et si une action innatendue apparaît, d'aider à expliquer pourquoi l'agent a décider d'exécuter cette action. Les similitudes avec notre approche viennent de l'utilisation de concepts d'agent. Les auteurs les utilisent dans le but de pouvoir facilement comparer des modèles de spécifications (diagramme d'état, de séquences...) au comportement réel des agents. Notre approche se distingue de la Tracing Method du fait de l'automatisation que nous proposons au niveau de l'instrumentation du code des agents, mais aussi au niveau de la détection des incohérences entre le comportement attendu et le comportement observé. Enfin, notre approche consiste à embarquer le contrôle au sein des agents pour permettre une détection dynamique des erreurs de comportement et non une étude post-mortem des traces de programme.

J.Vazquez-Salceda (Vasquez-Salceda *et al.*, 2004) propose un langage de normes, dans le cadre d'applications médicales, basé sur la logique déontique, permettant de détecter lorsqu'une norme n'est pas respectée et de punir l'agent en conséquence. Même si nos lois s'apparentent à des normes, leur finalité n'est pas d'orienter le comportement

des agents mais de vérifier qu'il correspond à des exigences. Contrairement à l'usage courant que l'on fait des normes, les agents ne décident pas de suivre ou non les lois, ils se doivent de les respecter. Les lois sont en quelque sorte des normes "enregimentées" ("enregimented norms" (Broersen *et al.*, 2003))

7 Conclusion

Nous avons présenté dans ce papier un système permettant la génération d'agents en mesure d'effectuer une vérification de leur propre comportement. Cette vérification consiste à s'assurer que les lois associées à un agent sont bien respectées tout au long de son exécution. Ces lois correspondent aux descriptions d'exigences sur le comportement et les états des agents. L'intérêt de notre approche est principalement de permettre à une personne extérieure au développement du système de vérifier qu'il respecte bien certaines règles mais aussi de simplifier le travail du développeur en lui épargnant l'instrumentation de son code. Enfin, un point important tient au fait que cette vérification des lois peut se faire sur n'importe quel agent et ce quelque soit le modèle utilisé pour l'implémenter. Ainsi, même un SMA aux agents hétérogènes peut être soumis à des lois.

La prochaine étape de nos travaux sera surtout axée sur la construction du langage de description des lois. Ce langage doit permettre de déduire, de façon générale, le code à insérer à l'intérieur des agents pour effectuer le contrôle du respect des lois, tout en étant assez complet pour pouvoir exprimer un grand nombre de type de lois. La génération du code à partir des lois pourrait passer par une étape de modélisation automatique des lois sous forme d'un réseau de Petri décrivant ce qui est interdit ou obligatoire pour un agent.

Références

BARBER K. S. & MARTIN C. E. (1999). Agent autonomy : Specification, measurement and dynamic adjustment. In *Proc. of the Autonomy Control Software workshop at Autonomous Agents'99*, p. 8–15.

BENERECETTI M., GIUNCHIGLIA F. & SERAFINI L. (1997). *Model Checking Multiagent Systems.* Rapport interne 9708-07, IRST, Povo, Italy.

BROERSEN J., DASTANI M. & VAN DER TORRE L. (2003). BDIO CTL, obligations and the specification of agent behavior. In *Proceedings of IJCAI'03*.

BURKHARD H. D. (1993). Liveness and fairness properties in multi-agent systems. In *Proceedings of Int. Joint Conferences on Artificial Intelligent* : IJCAI'93.

DE SOUSA DIAS M. & RICHARDSON D. J. (2002). *Issues on Software Monitoring*. Rapport interne, Department of Information and Computer Science, University of California, Irvine, CA.

EDMONDS B. & BRYSON J. J. (2004). The insufficiency of formal design methods - the necessity of an experimental approach for the understanding and control of complex MAS. In *Proceeding of AAMAS'04*, New York, USA.

FEATHER M. S., FICKAS S., VAN LAMSWEERDE A. & PONSARD C. (1998). Reconciling System Requirements and Runtime Behavior. In *Proceedings of IWSSD9*, Isobe, Japan.

HUANG Y. & KINTALA C. (1995). Software fault tolerance in the application layer. In *Software Fault Tolerance*, p. 231–248.

JOYCE J., LOMOW G., SLIND K. & UNGER B. (1987). Monitoring distributed systems. *ACM Transaction on Computer Science*, **5**(2), 121–150.

LAM D. N. & BARBER K. S. (2004). Verifying and explaining agent behavior in an implemented agent system. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2004)*, p. 1226–7, New York City.

LIAO Y. & COHEN D. (1992). A specificational approach ot high level program monitoring and measuring. *IEEE Trans. Software Engineering*, **18**(11).

LUMPP J. E., CASAVANT T. L., SIEGLE H. J. & MARINESCU D. C. (1990). Specification and identification of events for debugging and performance monitoring of distributed multiprocessor systems. In *Proceedings of the 10th International Conference on Distributed Systems*, p. 476–483.

MAHRENHOLZ D., SPINCZYK O. & SCHRÖDER-PREIKSCHAT W. (2002). Program instrumentation for debugging and monitoring with aspectC++. In *Proc. of the 5th IEEE International symposieum on Object-Oriented Real-time Distributed Computing*, Washington DC, USA.

MANSOURI-SAMANI M. (1995). *Monitoring of Distributed Sytems*. PhD thesis, University of London. Imperial College of Science Technology and Medecine, Departemnt of Computing, London, UK.

MARZILLO K., COOPER R., WOOD M. D. & BIRMAN K. P. (1991). Tools for distribution application management. *Cornell University, IEEE Computer*, p. 42–51.

SCHNOEBELEN P. (1999). *Vérification de Logiciels : Techniques et Outils du Model-Checking*. Vuibert edition.

VASQUEZ-SALCEDA J., ALDEWERLD H. & DIGNUM F. (2004). Implementing norms in multiagent systems. In *Proceedings of MATES'04*, Erfurt, Germany.

VON WRIGHT G. H. (1951). Deontic logic. *Mind*, **60**(237), 1–15.

WOOLDRIDGE M., FISHER M., HUGET M. P. & PARSONS S. (2002). Model checking multiagent systems with mable. In *Proceedings of AAMAS'02*, Bologna, Italy.

Partage de processeur pour agents autonomes conscients du temps ⋆

Cédric Dinont[1,2]

[1] Université des Sciences et Technologies de Lille
Laboratoire d'Informatique Fondamentale de Lille - UMR CNRS 8022
Equipe SMAC
59655 Villeneuve-d'Ascq Cedex

[2] Institut Supérieur de l'Electronique et du Numérique
41 Bd Vauban 59046 Lille Cedex
cedric.dinont@isen.fr

Résumé : Nous nous intéressons à la possibilité dans un système multi-agent de garantir le respect de contraintes temporelles du type dates limites de réponse. Ce problème que l'on rencontre dans les systèmes temps réel pose dans le cas des agents des difficultés supplémentaires. Les agents autonomes peuvent à tout instant décider de démarrer des traitements qui nécessitent, dans le cas d'agents cognitifs, beaucoup de temps processeur et ont des durées difficiles à prévoir. Ils doivent également pouvoir rester à l'écoute du monde extérieur pendant leur exécution pour prendre rapidement en compte les changements dans l'environnement. La conscience du temps est une propriété nécessaire lorsque l'on veut que ces agents puissent assurer des délais de traitement. En définissant une classe d'agents extravertis constamment tournés vers l'extérieur et une classe d'agents introvertis qui permettent de garantir des délais de traitement, nous montrons qu'il est possible pour des agents autonomes conscients du temps de cohabiter sur un même processeur. Nous introduisons également un Agent de Services Temporels (AST) qui collecte les requêtes d'utilisation du processeur et qui s'engage à ce qu'elles soient réalisées avant leur date limite.

Mots-clés : Systèmes multi-agents, Autonomie, Raisonnement temporel, Ordonnancement, Programmation par contraintes.

1 Introduction

1.1 Des agents cognitifs autonomes

Un des concepts fondamentaux apporté par le paradigme agent est celui d'autonomie (Joseph & Kawamura, 2001). Elle peut être définie comme la capacité dont dispose

⋆Ce travail est cofinancé par Thales Division Aéronautique et le Conseil Régional du Nord-Pas-de-Calais.

un agent de décider lui-même des actions qu'il entreprend. Le respect de contraintes temporelles dans un contexte d'autonomie pose de nombreux problèmes, que ce soit au niveau d'un agent particulier ou au niveau du système multi-agent (SMA) dans son ensemble. Nous nous situons dans le domaine de l'Intelligence Artificielle Distribuée (Sycara *et al.*, 1996). Les raisonnements de nos agents cognitifs peuvent mettre en jeu des algorithmes complexes pour résoudre des problèmes de planification par exemple. L'espace de recherche d'un problème peut présenter des îlots de complexité : pour une faible variation des données en entrée, le temps de calcul pour trouver la solution peut varier énormément, voire devenir prohibitif. Les ressources processeur nécessaires sont donc très variables et difficiles à prédire. De nombreux problèmes sont résolus par l'emploi d'heuristiques, plus ou moins performantes. Il est difficile de prédire si une heuristique donnera un résultat meilleur et/ou plus rapidement qu'une autre en fonction de la donnée en entrée. Quand le choix d'une heuristique particulière devient trop difficile, on en vient à en lancer plusieurs en parallèle. On les arrête dès qu'une d'entre elles trouve une solution. Notre proposition permet notamment de traiter ce genre de problèmes.

Nous attendons des agents qu'ils s'intéressent à leur environnement. Ils doivent être en permanence à l'écoute de l'extérieur pour prendre rapidement en compte les changements dans l'environnement et répondre aux sollicitations des autres agents. Pour des agents logiciels, cela revient à disposer régulièrement d'assez de temps processeur qu'ils utiliseront pour se remettre en phase avec leur environnement. Nous définissons les périodes d'activité et d'inactivité des agents et nous associons aux agents une horloge "temps agent" qui permet d'évaluer la quantité de travail qu'un agent a réalisé à un instant donné.

Définition 1 (Agent actif / agent inactif)
Un agent est inactif sur une période de temps donnée s'il ne dispose d'aucun accès processeur sur cette période. Il est actif sur la période considérée dans le cas contraire.

Définition 2 (Temps agent)
Le temps agent avance au rythme des opérations exécutées par l'agent.

Le temps agent est propre à chaque agent : il y a une horloge "temps agent" par agent qui peut avancer à un rythme différent pour chacun d'eux.

Comme nous venons de le voir, les traitements dans lesquels nos agents s'engagent ont des durées (en temps agent) qui ne sont pas forcément connues avant de les avoir réalisés. Un agent qui veut avoir la maîtrise de ce qu'il fait ne va pas se lancer dans l'exécution complète et en une seule fois de ses traitements. Il va plutôt décider de les commencer pendant un certain temps, de voir où il en est arrivé et d'éventuellement les continuer pendant une nouvelle période de temps. Il fera cela jusqu'à la terminaison de ses traitements ou jusqu'à ce qu'il renonce à les terminer.

Définition 3 (Traitement / tâche)
Un traitement est un processus de calcul. Une tâche est l'exécution d'un traitement pendant un temps agent fixé.

Un traitement peut être réalisé en une ou plusieurs tâches. Lorsqu'une tâche se termine, une nouvelle peut continuer le traitement où il en était arrivé.

1.2 Des agents conscients du temps

Les agents cognitifs raisonnent en permanence pour savoir quels buts atteindre et comment les atteindre. Le temps est un élément de la réalité dont les agents doivent avoir conscience s'ils veulent prendre les bonnes décisions.

Définition 4 (Agent conscient du temps)
Un agent est conscient du temps si son comportement dépend d'un temps qui s'écoule.

Il ne suffit pas qu'un agent raisonne sur le temps pour qu'il soit conscient du temps qui passe. Il faut de plus que ses raisonnements prennent en compte l'écoulement du temps pendant ses actions et prises de décision. La connaissance de l'écoulement du temps s'acquiert en consultant l'heure auprès d'une horloge. L'horloge à laquelle un agent conscient du temps se réfère n'a que peu d'importance. En particulier, il n'est pas nécessaire qu'elle soit la même pour les différents agents conscients du temps d'un même SMA.

Un agent conscient du temps pourra planifier des traitements, estimer la durée en temps agent nécessaire à leur exécution, lancer des tâches qui réalisent ces traitements, les surveiller et prendre des décisions en fonction de ses observations (Vincent *et al.*, 2001). Un agent qui décompose un traitement en plusieurs tâches va par exemple pouvoir détecter qu'un traitement dure anormalement longtemps et décider de le stopper définitivement. Il doit pouvoir continuer de fonctionner sans les données qu'il attendait en sortie d'un traitement s'il décide de le stopper en cours d'exécution. Remarquons enfin qu'il n'est pas nécessaire qu'un agent soit actif en permanence pour être conscient du temps.

La prise en compte de la durée des traitements est un problème récurrent en IA et dans les SMA (Garvey & Lesser, 1993). Les algorithmes anytime sont incrémentaux et proposent une fonction indiquant la qualité de la réponse en fonction du temps. Quand ils existent pour un problème donné, leur utilisation permet aux agents de raisonner sur les durées des tâches qu'ils allouent aux problèmes et même de s'imposer des contraintes sur la qualité de la solution. (Zilberstein & Mouaddib, 1999) proposent un niveau méta pour raisonner sur les incertitudes associées aux durées et aux qualités de réponse. (Adelantado & de Givry, 1995) ajoutent un mécanisme réactif aux agents anytime pour prendre en compte les besoins des applications temps réel embarquées. D'autres raisonnements peuvent être faits sur le nécessaire compromis entre temps de calcul et utilité de la réponse (Horvitz & Rutledge, 1991). (Wagner, 2000) détaille la proposition design-to-time et son extension design-to-criteria qui permettent de prendre en compte la durée des traitements et d'autres critères pour la planification. Enfin, (Prouskas, 2002) propose une extension d'April qui permet d'investiguer la différence qui existe entre les contraintes temporelles dans les relations entre un agent logiciel et un humain et les relations entre les agents. Les périodes de temps considérées lors de l'interaction avec un agent humain sont plus grandes qu'avec un agent logiciel. Les contraintes temps réel sont également moins fortes quand elles sont liées à l'agent humain.

Notons enfin qu'un processus temps réel s'exécutant sur un système temps réel n'est pas forcément conscient du temps. Il a juste été programmé pour donner une réponse dans un temps donné.

1.3 Partage d'un même processeur

Un des rôles du système d'exploitation est de gérer l'ordonnancement des tâches sur le ou les processeurs disponibles. Cela est fait différemment selon les propriétés recherchées. Ainsi, des systèmes comme Windows ou Unix, dits à temps partagé, garantissent que tous les processus pourront disposer régulièrement d'un quantum de temps processeur. Ce qui est recherché ici est l'illusion, au niveau d'abstraction de l'utilisateur, que tous les processus s'exécutent en même temps. Malheureusement, ces systèmes ne garantissent rien sur les dates de fin des tâches.

Cette garantie est en revanche fournie par les systèmes temps réels, mais cela est souvent obtenu au prix de la perte de l'illusion que les programmes sont toujours actifs. En effet, les algorithmes classiquement utilisés comme Earliest Deadline First (EDF) proposé par (Liu & Layland, 1973) préfèrent lancer une tâche jusqu'à ce qu'elle soit terminée et passer ensuite à la suivante plutôt que d'assigner de petits quantums de temps à chaque tâche.

La gestion des conflits d'allocation du processeur est également un point important qu'abordent notamment (DiPippo *et al.*, 2001). Dans un SMA, ces conflits seront de préférence gérés au niveau multi-agent. On peut ainsi utiliser des mécanismes d'interaction et de négociation pour les traiter en prenant en compte l'autonomie des agents. Si c'est le système d'exploitation qui décide de ce qu'il faut faire en cas de conflit, il prendra des décisions qui iront à l'encontre du principe d'autonomie que nous nous fixons. Par exemple, et c'est le cas le plus courant, considérons que le système d'exploitation ou l'infrastructure du SMA utilise des priorités pour gérer les conflits. Lorsqu'une inconsistance apparaîtra dans l'ordonnancement, on assistera à l'éviction autoritaire des tâches à priorité basse. Or, nous voulons que quand un agent décide de se lancer dans une tâche, il puisse avoir la garantie qu'elle se terminera comme il l'avait prévu. Il faut que cela soit valable pour tous les agents, sans distinction. Le principe même d'autonomie des agents va à l'encontre de la gestion des conflits par utilisation de priorités et soumission à l'autorité d'un tiers. En effet, deux agents peuvent penser à juste titre que leurs tâches sont très prioritaires. S'il existe un conflit pour l'exécution d'une nouvelle tâche, il est préférable qu'un agent décide lui-même de se sacrifier en stoppant une de ses tâches ou que ce soit la nouvelle tâche qui ne puisse pas s'exécuter si aucun agent ne se sacrifie.

En définitive, nous voulons des agents à l'écoute de l'extérieur et aussi pouvoir assurer des contraintes temporelles dans un contexte d'autonomie (Guessoum & Dojat, 1996) et de partage d'un unique processeur. Du fait de leur autonomie, les agents peuvent décider à tout instant de lancer une tâche et vouloir qu'elle soit terminée avant une date donnée. Il faut un moyen d'assurer que quand un agent exécute une tâche, les autres agents lui laissent la quantité de ressource processeur qui lui est nécessaire pour la finir avant sa date limite. C'est-à-dire qu'il faut un moyen de savoir si l'ensemble des contraintes temporelles liées aux tâches demandées est bien satisfiable. Il faut également pouvoir gérer les conflits qui surviennent quand une nouvelle requête de tâche apporte une inconsistance.

2 Architecture proposée

Les agents que nous considérons ne partagent pas de mémoire. Ils disposent chacun d'une boîte aux lettres et ne communiquent que de manière asynchrone par messages auxquels ils ne sont pas forcément tenus de répondre. Les changements dans l'environnement sont notifiés aux agents par messages. Nous choisissons pour la suite une granularité telle qu'un agent ne dispose que d'un flot d'exécution, c'est-à-dire qu'un agent peut s'exécuter dans un processus ou un thread, mais ne peut être constitué de plusieurs processus ou threads concurrents. Cela n'empêche pas un agent de disposer de différents contextes entre lesquels il commute.

2.1 Description générale

On classifie les agents en deux classes exclusives que nous définissons dans le paragraphe suivant : les agents extravertis et les agents introvertis. Ces classes sont caractérisées par le délai séparant deux consultations de leur boîte aux lettres. Les agents extravertis raisonnent sur le comportement temporel des agents introvertis et leur délèguent la réalisation des traitements longs sous forme de tâches. Ils disposent d'un lien qui leur permet de suspendre et de redémarrer à tout instant les agents introvertis qui travaillent pour eux.

L'Agent de Services Temporels (AST), décrit dans la section 3, est un agent extraverti particulier avec lequel les autres agents extravertis sont obligés de dialoguer lorsqu'ils veulent faire réaliser une tâche à un agent introverti. Il collecte les contraintes temporelles de toutes les tâches exécutées par les agents introvertis et s'engage à ce qu'elles soient vérifiées. En particulier, il trouve un ordonnancement pour les tâches et indique aux agents extravertis les créneaux temporels d'exécution de celles-ci. S'ils respectent leurs créneaux temporels, l'AST garantit que toutes les tâches se termineront avant leurs dates limites.

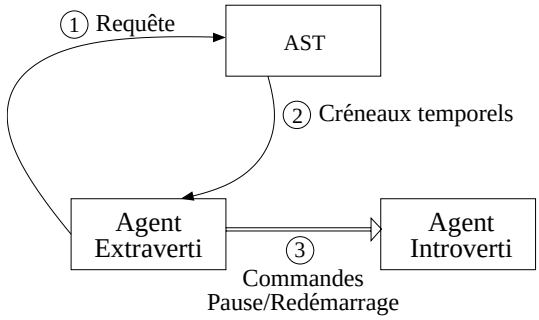


FIG. 1 – Relations principales entre l'AST, les agents extravertis et les agents introvertis.

2.2 Classes d'agents

Définition 5 (Agent extraverti / agent introverti)
Un agent est extraverti si le temps agent qui sépare deux consultations de sa boîte aux lettres est borné. Il est introverti dans le cas contraire.

Cette définition implique qu'un agent extraverti ne peut s'engager aveuglément dans des traitements longs qui l'empêcheraient de consulter régulièrement sa boîte aux lettres. Il est parfois possible de prévoir la consultation régulière des messages dans la boucle de traitement. Dans ce cas, un agent pourra lancer des traitements longs. Cela n'est pourtant pas possible dans tous les cas, notamment lors de l'utilisation de code hérité ou de traitements du type résolution de contraintes. Pour ce type de traitements, une fois lancés, il est quasi impossible de les suspendre le temps de consulter la boîte aux lettres et de les redémarrer ensuite où ils s'étaient arrêtés. Les agents extravertis délégueront l'exécution de ces traitements ininterruptibles à des agents introvertis.

Nous pouvons également remarquer qu'un agent extraverti ne peut pas avoir de période d'inactivité plus grande que la limite fixée entre deux consultations de la boîte aux lettres. Au contraire, un agent introverti peut avoir des périodes d'inactivité de durées illimitées. Nous nous intéressons dans la suite plus particulièrement aux agents extravertis actifs en permanence. Le temps partagé est la seule propriété nécessaire au niveau du système d'exploitation pour que des agents puissent être considérés comme constamment actifs. En effet, si on place tous les agents à la même priorité vis-à-vis de l'ordonnanceur du système d'exploitation, chaque agent disposera régulièrement d'un peu de temps processeur. Tout comme l'utilisateur d'un tel système considère que ses différents programmes fonctionnent en permanence, nous pouvons considérer que certains agents sont actifs en permanence. Cela n'a évidemment de sens que pour des périodes de temps considérées suffisamment grandes.

2.3 Respect des contraintes temporelles

La séparation en deux classes d'agents permet d'atteindre une partie du but que nous nous sommes fixé : exécuter des traitements longs tout en ayant des agents en permanence tournés vers l'extérieur. Cela ne suffit pas pour assurer que les traitements longs vérifient des contraintes temporelles comme des dates limites. Comme nous le verrons plus loin, la gestion des dates limites se fera au moyen d'un agent particulier qui indiquera quelles tâches doivent s'exécuter et quelles tâches doivent être en pause à un instant donné. Cela implique que les agents introvertis puissent être mis en pause et redémarrés à volonté. Le système de messagerie des agents ne peut être utilisé à cet effet, car une fois qu'un agent introverti s'est lancé dans l'exécution d'une tâche, il ne réagit plus aux messages qui s'accumulent dans sa boîte aux lettres. Un moyen supplémentaire devient donc nécessaire pour assurer la suspensibilité des agents.

Définition 6 (Suspensibilité)
Un agent est suspensible s'il est possible de lui ordonner à tout moment de devenir inactif ou de redevenir actif.

Un agent suspensible peut ne pas répondre à un message mais il est obligé de se suspendre quand on le lui indique. Quand un agent est suspendu, il est inactif et n'utilise donc plus du tout le processeur. Il n'est en particulier plus capable d'indiquer à l'agent extraverti pour lequel il travaille l'état d'avancement de son traitement. Si la connaissance de cet état d'avancement est nécessaire aux raisonnements temporels envisagés par l'agent extraverti et que l'agent introverti dispose d'un moyen de le déterminer, il faut qu'il profite de ses moments de conscience pour le lui communiquer.

Nous remarquons que les agents introvertis doivent être suspensibles pour qu'une gestion de dates limites puisse être mise en place. Il faut accepter que des agents perdent une certaine part de leur autonomie, ici en acceptant de devenir à tout moment inactifs, pour que les autres agents puissent s'exécuter et ainsi respecter leurs délais. Cette propriété n'est pas nécessaire pour tous les agents. Elle ne peut pas être appliquée aux agents extravertis actifs en permanence que nous avons introduits précédemment car ils ne seraient plus constamment actifs.

Nous remarquons enfin que les propriétés des agents décrites jusqu'ici (actif, conscient du temps, extraverti et suspensible) sont intrinsèques à l'agent et indépendantes de l'environnement d'exécution.

3 L'Agent de Services Temporels

Nous avons vu que chaque agent dispose d'une horloge personnelle qui lui indique son "temps agent". Ils doivent également se référer dans leurs raisonnements à une horloge, quelconque, pour être conscients du temps. L'AST collecte les dates limites des tâches des différents agents. Ces dates doivent toutes se référer à une horloge commune. Nous définissons à cet effet le "temps AST".

Définition 7 (Temps AST)
Le temps AST est un temps commun à tous les agents qui utilisent les services de celui-ci.

On choisira souvent l'horloge système pour donner le temps AST, mais il est possible que ce soit une horloge virtuelle qui le donne, par exemple dans le cas de simulations.

3.1 Protocole de requête de tâches

Un agent qui veut allouer une tâche pour faire avancer un traitement fait une requête à l'AST sous la forme $[ET,\ CTs]$ où ET est le temps agent demandé pour la réalisation de la tâche et CTs une liste de contraintes temporelles. L'AST évalue si l'introduction des contraintes temporelles de la nouvelle tâche ne provoque pas d'inconsistance. Si une inconsistance est détectée, l'AST renvoie à l'agent demandeur un refus d'exécution de la tâche. Ce dernier peut formuler une nouvelle requête s'il le souhaite. Si la requête ne provoque pas d'inconsistance, l'AST renvoie à l'agent demandeur la liste des créneaux temporels (en temps AST) pendant lesquels la tâche peut s'exécuter. L'agent demandeur doit respecter ces créneaux en envoyant des commandes de suspension et de

redémarrage vers l'agent qui va exécuter la tâche. Un agent peut indiquer à tout moment à l'AST qu'il renonce à utiliser certains des créneaux temporels qui lui ont été attribués. Cela peut être nécessaire lorsque le traitement que la tâche réalise se termine avant la fin de la période de temps allouée à la tâche.

3.2 Ordonnancement

3.2.1 Propriétés recherchées

L'ordonnanceur dont nous avons besoin doit disposer de propriétés particulières. En se plaçant au-dessus de celui du système d'exploitation, il peut prendre en compte le fait qu'à son niveau, plusieurs tâches peuvent s'exécuter en parallèle. Notamment, il doit prendre en compte de manière particulière les agents extravertis qui sont toujours actifs. Il doit pouvoir gérer des dates limites, tout en étant facilement extensible à d'autres contraintes temporelles.

Nous considérons qu'il est intéressant de commencer les tâches au plus tôt. Le comportement d'un ordonnanceur comme EDF, qui va exécuter entièrement la tâche dont la date limite est la plus proche puis entièrement la tâche suivante, n'est notamment pas adapté à la résolution d'un même problème par différents agents avec différentes heuristiques. Dans ce genre d'application, il est intéressant de démarrer au plus tôt les différentes heuristiques et d'arrêter la résolution dès qu'un agent a trouvé la solution recherchée.

3.2.2 Modélisation

Les requêtes sont sous la forme $[ET_i,\ DL_i]$. ET_i est le temps agent à affecter à la tâche i avant la date limite DL_i. On parlera également pour ET_i d'énergie de calcul totale à attribuer à la tâche i. On considère n requêtes dont les dates limites sont triées par ordre croissant : $DL_{i+1} \geq DL_i, \quad 1 \leq i \leq n-1$. On obtient n intervalles : [T_0 à DL_1, DL_1 à DL_2, ...,DL_{n-1} à DL_n], avec T_0 la date de début de l'ordonnancement. On considère également un nombre NAE fixe d'agents extravertis actifs en permanence.

Nous voulons respecter les dates limites : pour le $i^{ème}$ intervalle, le but est donc de terminer la $i^{ème}$ tâche avant la fin de l'intervalle. Nous voulons aussi donner le plus possible de temps processeur aux tâches $i+1$ à n et ceci le plus tôt possible dans l'intervalle. On partitionne l'intervalle considéré en sous-intervalles correspondants à toutes les possibilités de faire tourner la tâche i en parallèle d'une ou plusieurs autres des tâches $i+1$ à n. Pour le $i^{ème}$ intervalle, il y a 2^{n-i} sous-intervalles. On ordonne les sous-intervalles par ordre décroissant du nombre de tâches en faisant partie. Pour un sous-intervalle comportant T tâches, chaque tâche dispose d'une part $P_{i,j}$ de la puissance processeur disponible égale à $\frac{1}{T+NAE}$.

Ce partitionnement d'un intervalle i correspond bien à ce que nous voulons faire : la priorité est donnée à la terminaison de la tâche i puisqu'elle apparaît dans tous les sous-intervalles et si on peut exécuter une ou plusieurs autres tâches en parallèle de la tâche i, on le fera le plus tôt possible dans l'intervalle. Le problème pour le $i^{ème}$ intervalle

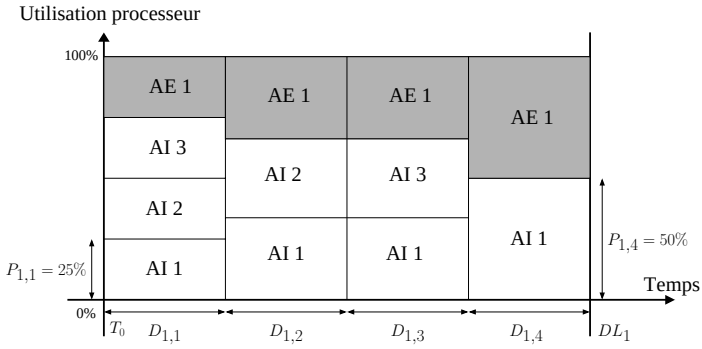


FIG. 2 – Partitionnement du premier intervalle dans le cas de 3 agents introvertis et d'1 agent extraverti.

revient donc à déterminer les durées $D_{i,j}$, $D_{i,j}$ étant la durée du sous-intervalle j de l'intervalle i. Les $D_{i,j}$ peuvent s'annuler.

3.2.3 Préparation de la résolution

La résolution est faite intervalle par intervalle. On calcule pour chaque intervalle les variables nécessaires à la constitution d'un système d'équations et d'inéquations linéaires que l'on résout grâce à l'algorithme du simplexe (Faure, 1979).

On définit tout d'abord les limites de l'intervalle considéré :

– la date de fin est toujours égale à DL_i,
– la date effective de début DD_i peut être différente de DL_{i-1} si la tâche $i-1$ s'est terminée avant la fin de son intervalle. Elle est obtenue par :

$$DD_1 = T_0 \qquad \text{et} \qquad DD_i = \sum_{j=1}^{2^{n-i-1}} D_{i-1,j}, \quad 1 \leq j \leq 2^{n-1}$$

Soit $L_{i,j,k}$ qui vaut 1 si la tâche k possède une de ses parties dans le sous-intervalle j de l'intervalle i et 0 sinon. Et soit $E_{i,k}$ l'énergie totale effectivement affectée à la tâche k dans l'intervalle i.

$$E_{i,k} = \sum_{j=1}^{2^{n-k}} L_{i,j,k} P_{i,j} D_{i,j}$$

On peut calculer $ER_{i,k}$, l'énergie restante pour la tâche k à partir de l'intervalle i inclus, ainsi :

$$ER_{1,k} = ET_k \qquad \text{et} \qquad ER_{i,k} = ER_{i-1,k} - E_{i-1,k}, \quad 2 \leq i \leq n, \quad 1 \leq k \leq 2^{n-1}$$

Comme nous résolvons le problème intervalle par intervalle, nous devons nous assurer que nous choisissons correctement les tâches qui vont s'exécuter dans un intervalle i particulier. En effet, pour les tâches qui ne pourraient pas s'exécuter entièrement dans

leur intervalle, nous devons prendre en compte dans les intervalles précédents que l'on doit absolument leur affecter une part de l'énergie disponible. Un exemple dans le paragraphe 3.2.5 illustre cela.

Soit ESR_i l'énergie minimale à affecter dans l'intervalle i à d'autres tâches que la $i^{ème}$. Et soit $C_{i,k}, \quad i+1 \leq k \leq 2^{n-1}$ qui vaut 1 si la tâche k fait partie de la liste des tâches pour lesquelles il faut affecter un minimum d'énergie dans l'intervalle i. Le calcul des $C_{i,k}$ et de ESR_i se fait en partant du dernier intervalle et en remontant vers l'intervalle i. On initialise tout d'abord ESR_i et les $C_{i,k}$ à 0. A l'intervalle m, on met à jour ESR_i et les $C_{i,k}$ de la manière suivante :

- Soit M_m l'énergie processeur disponible dans l'intervalle m.
  $M_m = P(DL_m - DD_m)$ avec P la puissance processeur par unité de temps.
- Si $ER_{i,m} + ESR_i \leq M_m$, alors on réinitialise ESR_i et les $C_{i,k}$ à 0.
- Sinon, on ajoute $(ER_{i,m} - M_m)$ à ESR_i et on place $C_{i,m}$ à la valeur 1.

3.2.4 Résolution

Nous avons maintenant déterminé toutes les valeurs numériques utiles et il ne reste plus que les $D_{i,j}$ comme variables à déterminer. Le programme linéaire à résoudre pour cela est le suivant :

$$Max\ z = \sum_{j=1}^{2^{n-i}} [(2^{n-i} - j + 1)D_{i,j}] \tag{1}$$

$$D_{i,j} \geq 0, \quad 1 \leq j \leq 2^{n-i} \tag{2}$$

$$\sum_{j=1}^{2^{n-i}} D_{i,j} \leq DL_i - DD_i \tag{3}$$

$$E_{i,i} = ER_{i,i} \tag{4}$$

$$E_{i,k} \leq ER_{i,k}, \quad j+1 \leq k \leq 2^{n-1} \tag{5}$$

$$\sum_{k=j+1}^{2^{n-1}} (C_{i,k}E_{i,k}) \geq ESR_i \tag{6}$$

(1) On cherche à maximiser la somme pondérée des durées $D_{i,j}$ de l'intervalle i. Les facteurs de pondération sont tels que le facteur de $D_{i,j}$ soit plus grand que le facteur de $D_{i,j+1}$.

(2) Toutes les durées sont positives.

(3) La tâche i doit se terminer avant sa date limite : la somme des durées de ses parties est inférieure ou égale à la durée de l'intervalle courant.

(4) La somme des énergies des parties de la tâche i dans l'intervalle i doit être égale à l'énergie restante à affecter pour la tâche i.

(5) On ne peut affecter plus d'énergie à une tâche dans l'intervalle i que la quantité qui reste pour cette tâche à partir de cet intervalle.

(6) Il faut absolument consacrer la quantité ESR_i de l'énergie disponible dans l'intervalle courant pour les tâches k dont la variable $C_{i,k}$ vaut 1.

3.2.5 Exemples de résolution

Nous considérons pour ce paragraphe que le processeur peut exécuter 1000 opérations par seconde. Les requêtes de tâches $[ET, DL]$ sont formulées ainsi :

– ET est une durée en temps agent donnée en nombre d'instructions à réaliser,

– DL est la date limite pour la tâche en temps AST donnée en secondes.

Notre premier exemple met en jeu trois agents introvertis et aucun agent extraverti. Les requêtes sont : $[3000ops, 4s]$, $[2700ops, 6s]$ et $[2000ops, 8s]$. Nous remarquons qu'il est impossible d'exécuter entièrement la tâche 2 uniquement dans le second intervalle. Celui-ci a une durée de deux secondes et donc une capacité d'exécution de seulement 2000 opérations. Il faut exécuter au moins 700 opérations de la tâche 2 dans le premier intervalle. La figure 3 illustre cela : on doit arrêter la tâche 3 qui tournait en parallèle de 1 et 2 pour que la somme des parties de la tâche 2 dans le premier intervalle puisse atteindre la valeur de 700 opérations.

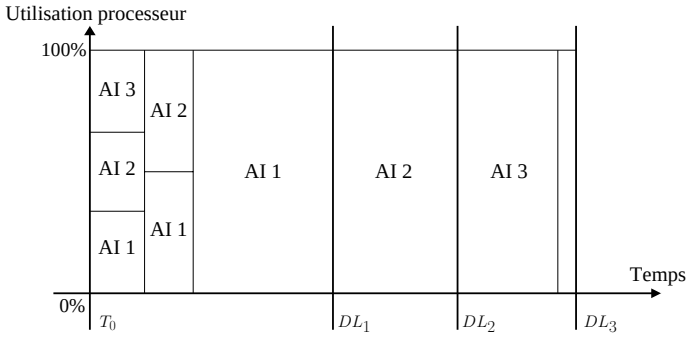


FIG. 3 – Prise en compte dans la résolution pour un intervalle de contraintes sur les intervalles restants.

Prenons, pour notre second exemple, deux agents extravertis et les requêtes suivantes pour trois agents introvertis : $[700ops, 4s]$, $[1000ops, 6s]$ et $[1300ops, 8s]$. La figure 4 montre que l'on arrive à terminer la tâche 1 avant la fin de son intervalle tout en exécutant les tâches 2 et 3 en parallèle. La nouvelle date de début du second intervalle n'est plus DL_1, mais la somme des $D_{1,j}$. Il en est de même pour la date de début du troisième intervalle puisque la tâche 2 se termine avant la date DL_2.

3.3 Implémentation

Nos agents s'exécutent chacun dans un processus système distinct. Le réveil des agents extravertis aux dates limites des créneaux temporels qui leur sont attribués ainsi que l'envoi des commandes de suspension et de réveil des agents introvertis ont été implémentés à l'aide de signaux système (norme POSIX).

Nous avons étudié les limites de l'ordonnanceur. La complexité provient principalement du découpage en sous-intervalles, qui est en $O(2^n)$. Le temps de calcul devient très vite prohibitif. Il est cependant possible en pratique de faire baisser cette complexité

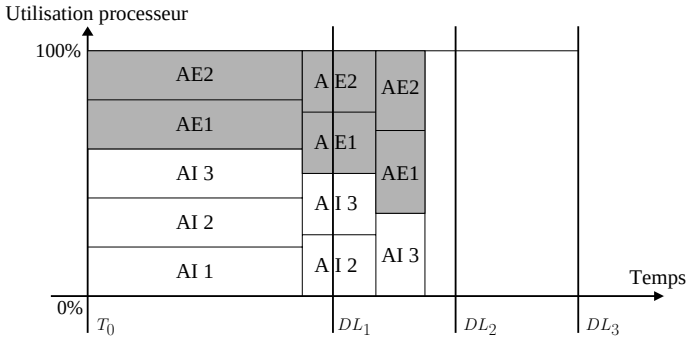


FIG. 4 – Modification des dates de début d'intervalles.

en étudiant les contraintes posées. En effet, il est souvent possible de prédéterminer, après le calcul de ESR_i et des $C_{i,k}$ qu'un certain nombre de tâches ne pourront pas du tout s'exécuter dans l'intervalle i considéré. Si la somme $ESR_i + ER_{i,i}$ est égale à la durée de l'intervalle i, il n'est pas nécessaire de considérer d'autres tâches que la tâche i et les tâches k pour lesquelles $C_{i,k} = 1$. De plus, il est nécessaire de limiter le nombre de tâches dans un intervalle pour que la solution trouvée dans celui-ci n'apporte pas un trop grand fractionnement. Il doit subsister une différence entre les ordres de grandeur des périodes de temps considérées par l'ordonnanceur système et celui de l'AST. Ce dernier ne doit pas empiéter sur le travail du premier en mettant en pause et en redémarrant trop souvent les agents.

L'implémentation gère, comme présenté plus haut, un nombre d'agents extravertis fixe et un nombre d'agents introvertis évolutif. Il est possible d'étendre le système pour prendre en compte un nombre d'agents extravertis variable. Il faudra cependant l'autorisation de l'AST pour que les nouveaux agents démarrent à un moment où ils ne perturbent pas les engagements pris auparavant.

Dans l'algorithme, nous offrons à tous les agents une part de $\frac{1}{nai+NAE}$ du processeur, avec nai le nombre d'agents extravertis qui s'exécutent à l'instant considéré. L'hypothèse que les agents extravertis utiliseront entièrement toute la capacité de calcul qui leur est offerte est trop forte quand ils se limitent à des automates de traitement des messages entrants. Le changement d'hypothèse sur l'utilisation processeur des agents extravertis se fait très simplement au moment de l'attribution des $P_{i,j}$.

4 Conclusion

Nous donnons une définition d'agent conscient du temps. Nous proposons une architecture pour que des agents cognitifs autonomes conscients du temps puissent partager un même processeur. Des agents extravertis ayant les propriétés d'être constamment actifs et tournés vers le monde extérieur délèguent les traitements longs aux durées connues, inconnues ou estimées pour lesquels ils veulent assurer des contraintes temporelles à des agents introvertis sous forme de tâches aux durées fixées. Les agents

introvertis exécutent leurs tâches avant de s'intéresser au monde extérieur. Ils acceptent également d'être suspendus à tout moment pour que les autres agents introvertis puissent s'exécuter eux aussi. La garantie de la satisfiabilité des contraintes temporelles pour l'ensemble des agents partageant un même processeur incombe à un agent extraverti particulier : l'Agent de Services Temporels. Nous définissons le protocole de communication utilisé pour dialoguer avec celui-ci, ainsi qu'un algorithme qui permet de vérifier la satisfiabilité du système de contraintes et d'ordonnancer les tâches.

5 Travaux futurs

La conscience du temps implique très rapidement la capacité à estimer les durées des traitements que l'on planifie. Si l'on veut pouvoir implanter dans les agents un large spectre d'algorithmes, d'IA ou non, il faut disposer d'un évaluateur des durées de traitement générique et certainement rétro-actif pour qu'il puisse améliorer ses estimations au cours du temps.

Le protocole que nous proposons renvoie un message de refus lorsqu'une inconsistance est détectée dans l'ensemble des contraintes temporelles posées. Une extension serait qu'il renvoie plutôt une liste de propositions de modification à apporter aux différentes requêtes de tâches précédemment postées. Ainsi, des interactions ou négociations pourront être ouvertes avec les autres agents comme dans (Garvey *et al.*, 1994) pour leur demander de renoncer à certaines de leurs exigences et ainsi laisser les nouvelles tâches s'exécuter.

Nous proposons un système qui permet le partage d'un unique processeur par plusieurs agents. La distribution de notre système pourra par exemple faire intervenir un AST par processeur et des interactions entre les ASTs pour l'équilibrage de charge par migration d'agents.

Il serait enfin intéressant d'imaginer qu'un agent puisse passer de l'état extraverti à l'état introverti et inversement. Le second passage n'est pas trivial : si le traitement réalisé en mode introverti n'est pas terminé au moment où la tâche se termine, il y a une mémorisation de contexte à faire avant de pouvoir repasser en mode extraverti.

Références

ADELANTADO M. & DE GIVRY S. (1995). Reactive/anytime agents - towards intelligent agents with real-time performance. In *IJCAI'95 Workshop on Anytime Algorithms and Deliberation Scheduling*.

DIPIPPO L. C., WOLFE V. F., NAIR L., HODYS E. & UVAROV O. (2001). A real-time multi-agent system architecture for e-commerce applications. In *ISADS*, p. 357–364.

FAURE R. (1979). *Précis de recherche opérationnelle*. Dunod, 4e edition.

GARVEY A., DECKER K. & LESSER V. (1994). *A Negotiation-based Interface Between a Real-time Scheduler and a Decision-Maker*. Rapport interne UM-CS-1994-008.

GARVEY A. & LESSER V. (1993). *A Survey of Research in Deliberative Real-Time Artificial Intelligence*. Rapport interne.

GUESSOUM Z. & DOJAT M. (1996). A real-time agent model in an asynchronous-object environment. In R. VAN HOE, Ed., *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*.

HORVITZ E. & RUTLEDGE G. (1991). Time-dependent utility and action under uncertainty. In *Proceedings of the seventh conference (1991) on Uncertainty in artificial intelligence*, p. 151–158 : Morgan Kaufmann Publishers Inc.

JOSEPH S. & KAWAMURA T. (2001). *Agent Engineering*, chapter Why Autonomy Makes the Agent. World Scientific Publishing.

LIU C. L. & LAYLAND J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, **20**(1), 46–61.

PROUSKAS K. (2002). *Real-Time Extensions of April for Time-Aware Multi-Agent Systems Programming*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London.

SYCARA K. P., PANNU A., WILLIAMSON M., ZENG D. & DECKER K. (1996). Distributed intelligent agents. *IEEE Expert*, **11**(6), 36–46.

VINCENT R., HORLING B., LESSER V. & WAGNER T. (2001). Implementing Soft Real-Time Agent Control. *Proceedings of the 5th International Conference on Autonomous Agents*, p. 355–362.

WAGNER T. (2000). *Toward Quantified, Organizationally Centered, Decision Making and Coordination*. PhD thesis, University of Massachusetts.

ZILBERSTEIN S. & MOUADDIB A. I. (1999). Reactive control of dynamic progressive processing. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*.

Un cadre théorique et pratique commun aux formalismes SAT et CSP n-aires.

Lionel Paris[1,2], Belaïd Benhamou[1,3], Pierre Siegel[1,4]

[1] Équipe Inférence Contraintes et Application,
Laboratoire des Sciences de l'Information et des Systèmes,
CMI, Université de Provence,
Technopôle de Château Gombert,
39, rue Joliot Curie, 13453 Marseille Cedex 13
http://www.lsis.org et http://www.cmi.univ-mrs.fr

[2] Lionel.Paris@cmi.univ-mrs.fr et
http://www.cmi.univ-mrs.fr/∼lparis

[3] Belaid.Benhamou@cmi.univ-mrs.fr

[4] Pierre.Siegel@cmi.univ-mrs.fr

Résumé : Dans ce papier, nous étudions une généralisation de la forme *CNF* qui permet de coder efficacement les CSP n-aires sous une forme booléenne. Nous montrons que la complexité en espace de ce codage est identique à celle de la représentation sous forme de CSP. Nous introduisons une nouvelle règle d'inférence, dont la saturation équivaut à l'application de la consistance d'arc (en temps linéaire) pour les CSP n-aires exprimés dans ce codage booléen. Deux méthodes énumératives pour la résolution de problèmes ainsi exprimés sont proposées : la première (équivalente à *MAC* dans les CSP) maintient la propriété de consistance d'arc à chaque noeud de l'arbre de recherche, tandis que la seconde (équivalente à *FC*) ne maintient qu'une forme partielle de consistance d'arc à chaque noeud. Ces deux méthodes sont expérimentées et comparées sur des instances du problème de Ramsey, ainsi que sur des CSP d'arités 3/4 générés aléatoirement. Des résultats satisfaisants sont obtenus.

Mots-clés : Satisfaction de contraintes, satisfiabilité.

1 Introduction

La résolution de contraintes est un domaine bien connu de l'intelligence artificielle. Généralement, deux formalismes sont utilisées : le calcul propositionnel, le plus souvent sous forme CNF et associé au problème (*SAT*) et le formalisme des problèmes de satisfactions de contraintes discrets (*CSP*).

Les méthodes de résolutions du problème SAT en calcul propositionnel sont plus anciennes que celles des CSP. Elles ont été largement étudiées depuis l'introduction de

la procédure de *Davis et Putnam* (DP) (Davis & Putnam, 1960), et sont toujours un domaine d'investigation pour une large communauté de chercheurs. Plusieurs améliorations de la procédure DP ont été proposées : tout d'abords *DPLL* (Davis *et al.*, 1962) et, plus récemment, (Silva & Sakallah, 1996; Malik *et al.*, 2001; Goldberg & Novikov, 2002). Ces nouvelles méthodes sont capables de résoudre un panel important d'instances SAT et sont utilisées pour résoudre des applications réelles. L'avantage des méthodes SAT est leur flexibilité, permettant de résoudre n'importe quelle contrainte exprimées sous forme *CNF*[1] (elles ne sont pas sensibles à l'arité[2] des contraintes comme c'est souvent le cas pour les méthodes des CSP). D'un autre côté, la formulation CNF perd la structure du problème qui est utilisée par les algorithmes dans le formalisme CSP.

Le formalisme des CSP discrets a été introduit en 1974 par Montanari (Montanari, 1974). L'avantage de ce formalisme est sa capacité à représenter explicitement la structure du problème exprimé à l'aide d'un graphe ou d'un hyper-graphe. Cette structure aide la propagation des contraintes et fournit des heuristiques qui rendent la résolution plus efficace. Un problème qui se pose est que les méthodes de résolution des CSP sont sensibles à l'arité des contraintes ; la plupart des méthodes connues s'appliquent aux CSP binaires[3]. Il en résulte une restriction car la plupart des problème réels (voir CSPlib[4]) s'expriment naturellement à l'aide de contraintes quelconques. Pour pallier cette restriction, des travaux récents fournissent des méthodes pour traiter des contraintes plus générales (Bessière *et al.*, 2002).

Les formalismes SAT et CSP sont étroitement liés. Il existe plusieurs travaux sur les transformations de CSP binaires en instances SAT (Kleer, 1989; Rossi *et al.*, 1990). La transformation décrite dans (Kasif, 1990) est généralisée aux CSP non binaires dans (Bessière *et al.*, 2003). Les différentes transformations de la forme SAT à la forme CSP sont décrites dans (Bennaceur, 1996; Walsh, 2000; Russel, 2004). Mais, la plupart des transformations de CSP vers SAT souffrent d'un accroissement de la taille du problème et de la perte de sa structure. Notre but est de fournir une représentation booléenne plus générale incluant les deux formalismes et conservant leurs avantages ; c'est à dire une représentation qui n'augmente pas la taille du problème et qui en conserve la structure. Nous montrons en particulier comment les CSP non binaires sont bien exprimés et efficacement résolus dans ce nouveau formalisme. Nous proposons deux méthodes énumératives pour cette formulation booléenne généralisée qui sont basée sur la procédure DPLL. Nous implémentons une nouvelle règle d'inférence qui utilise la structure du probème, pour effectuer la propagation de contrainte. L'application de cette règle au problème considéré est effectuée avec une complexité en temps linéaire. Nous montrons que la saturation de cette règle est équivalente à l'application de la consistance d'arc sur le CSP correspondant. Nous obtenons ainsi une bonne complexité en temps pour appliquer la consistance d'arc sur les CSP non binaires. Ceci nous permet de maintenir efficacement la consistance d'arc à chaque noeud de l'arbre de recherche dans une première méthode de résolution ; ceci est équivalent à la méthode *MAC* (D. Sabin, 1997)

[1] Une formule CNF est une conjonction de disjonction de littéraux.

[2] L'arité d'une contrainte dans un CSP est le nombre de variables qui y sont impliquées.

[3] Un CSP binaire ne contient que des contraintes d'arité 2.

[4] http ://csplib.org

pour les CSP. Une exploitation partielle de la règle d'inférence mène à une seconde méthode énumérative, qui est équivalente à la méthode Forward Checking (*FC*) pour les CSP non-binaires.

La suite de ce papier est organisé comme suit : dans la section 2 nous rappelons quelques définitions sur le calcul propositionnel et le formalisme des CSP discrets. Nous introduisons dans la section 3 une forme normale généralisée que nous utilisons pour exprimer à la fois les problèmes SAT et CSP. Nous définissons dans la section 4 une nouvelle règle d'inférence que nous utilisons pour montrer des résultats sur la consistance d'arc. Nous décrivons dans la section 5 les deux algorithmes énumératifs (les versions FC et MAC) basées sur deux différentes utilisations de la règle d'inférence. Des expérimentations sur des problème générés alléatoirement, ainsi que sur des instances du problème de Ramsey sont menées dans la section 6 pour montrer l'efficacité de nos méthodes de résolutions. La section 7 résume divers travaux existants en relation avec le notre et la section 8 conclut ce travail et donne quelques perspectives.

2 Préliminaires

Une formule CNF f, en logique propositionnelle, est une conjonction $f = C_1 \wedge C_2 \dots C_k$ de clauses. Chaque clause C_i est une disjonction de littéraux, $C_i = l_1 \vee l_2 \dots l_m$ où chaque littéral l_i est une occurence d'une variable booléenne soit sous sa forme positive, soit sous sa forme négative. Une interprétation I est une fonction qui assigne à chaque variable booléenne la valeur *vrai* ou *faux*. Une clause est satisfaite si on donne la valeur *vrai* à au moins un de ses littéraux l_i dans l'interprétation I ($I[l_i] =$ *vrai*). La clause vide est insatisfiable et est représentée par $\square$. La formule f est satisfaite par I si toutes ses clauses sont satisfaites par I, dans ce cas, I est un *modèle* de f. Une formule est satisfiable si elle possède au moins un modèle, sinon elle est insatisfiable.

D'un autre côté un CSP est un quadruplet $P = (X, D, C, R)$ où $X = \{X_1, X_2, \dots, X_n\}$ est un ensemble de n variables, $D = \{D_1, D_2, \dots, D_n\}$ est un ensemble de domaines finis discrets (D_i est le domaine des valeurs possibles pour X_i). $C = \{C_1, C_2, \dots, C_m\}$ est un ensemble de m contraintes, où la contrainte C_i est définie sur un sous-ensemble de variables $\{X_{i_1}, X_{i_2}, \dots, X_{i_{a_i}}\} \subset X$. L'arité de la contrainte C_i est a_i et $R = \{R_1, R_2, \dots, R_m\}$ est un ensemble de m relations, où R_i est la relation correspondante à la contrainte C_i. R_i contient les combinaisons de valeurs permises (les tuples) pour les variables impliquées dans la contrainte C_i. Un CSP binaire est un CSP dont les contraintes sont toutes d'arité deux (contraintes binaires). Un CSP est non-binaire si il contient au moins une contrainte dont l'arité est supérieure à 2 (une contrainte n-aire). Une *instantiation* I est une affectation qui assigne à chaque variable X_i une valeur de son domaine D_i. Une contrainte C_i est satisfaite par une instantiation I si la projection de I sur les variables impliquées dans C_i est un tuple de la relation R_i. Une instantiation I d'un CSP P est consistante (ou est une solution de P) si elle satisfait toutes les contraintes de P. Un CSP P est consistant si il admet au moins une solution ; sinon il est inconsistant. SAT et CSP sont tous deux des problèmes NP-complet Dans la suite du papier, on considère que n est le nombre de variables du CSP, m est son nombre de contraintes, a l'arité maximale de ses contraintes et d la taille du plus grand de ses domaines.

3 Un codage commun pour CNF et CSP

L'idée d'encoder un CSP sous la forme SAT fut introduite par De Kleer dans (Kleer, 1989). Il proposa le *codage direct*. Ensuite, Kasif (Kasif, 1990) a proposé le *codage AC* pour les CSP binaires. Plus récemment, Bessière et al. (Bessière *et al.*, 2003) ont généralisé le *codage AC* aux CSP n-aires. Notre approche, différente, consiste à fournir une forme booléenne généralisée qui peut prendre en compte les formes CNF (SAT) et CSP, au lieu de transformer des CSP sous la forme SAT. Nous décrivons dans cette section ce nouveau codage booléen et nous montrons comment les CSP n-aires sont naturellement représentés de manière optimale (sans accroissement de la taille de la représentation par rapport à la formulation CSP) dans ce codage.

3.1 La forme normale généralisée (GNF)

Une clause généralisée C est une disjonction de formules booléennes $f_1 \vee \ldots \vee f_m$ où chaque f_i est une conjonction de littéraux, *i.e.* $f_i = l_1 \wedge l_2 \wedge \ldots \wedge l_n$. Une formule est sous forme normale généralisée (GNF) si et seulement si c'est une conjonction de clauses généralisée. La sémantique des clauses généralisée est triviale : la clause généralisée C est satisfaite par une interprétation I si au moins une de ses conjonctions f_i ($i \in [1, m]$) est *vrai* dans I, sinon elle est falsifiée par I. Une clause classique est une clause généralisée simplifiée dans laquelle toutes les conjonctions f_i sont réduites à de simples littéraux. Ceci prouve que GNF est une généralisation de CNF. Nous montrons dans la suite que chaque contrainte C_i d'un CSP donné est représentable par une clause généralisée. Nous obtenons la taille de la représentation optimale en utilisant les formules de cardinalité $(\pm 1, L)$ qui signifie "exactement 1 littéral parmi ceux de la liste L doit être affecté à *vrai* dans tout modèle", pour exprimer efficacement que chaque variable d'un CSP ne doit être affectée qu'à une unique valeur de son domaine. On note *CGNF* la forme *GNF* combinée aux formules de cardinalité.

3.2 Le codage CGNF pour les CSP n-aires

Soit un CSP n-aire $P = (X, D, C, R)$. Nous définissons l'ensemble des variables booléennes utilisées pour la représentation, puis deux types de clauses : les *clauses de domaine* et les *clauses de contrainte*, nécessaires pour encoder les domaines et les contraintes du CSP.

- L'ensemble des variables booléennes : comme dans les autres codages on associe une variable booléenne Y_v à chaque valeur possible v du domaine de chaque variable Y du CSP. Ainsi, Y_v =*vrai* signifie que la valeur v est affectée à la variable Y du CSP. Nous avons besoin de $\sum_{i=1}^{n} |D_i|$ variables booléennes. Ce nombre est majoré par nd.
- Les clauses de domaine : soient Y une variable CSP et $D_Y = \{v_0, v_1, \ldots, v_k\}$ son domaine. La formule de cardinalité $(\pm 1, Y_{v_0} \ldots Y_{v_k})$ oblige la variable Y à être affecté à exactement une valeur de D_Y. Nous avons besoin de n formules de cardinalité pour représenter les n clauses de domaine.
- Les clauses de contrainte : chaque contrainte C_i du CSP P est représentée par la clause généralisée définie comme suit : Soit R_i la relation correspondant à C_i im-

pliquant l'ensemble de variables $\{X_{i_1}, X_{i_2}, \ldots, X_{i_a}\}$. Chaque tuple $t_j = (v_{j_1}, v_{j_2}, \ldots, v_{j_a})$ de R_i est exprimé par la conjonction $f_j = X_{v_{j_1}} \wedge X_{v_{j_2}} \wedge \ldots \wedge X_{v_{j_a}}$. Si R_i contient k tuples $t_1, t_2, \ldots, t_k$, on introduit la clause généralisée $C_{C_i} = f_1 \vee f_2 \vee \ldots \vee f_k$, pour exprimer la contrainte C_i. Nous avons besoin de m clauses généralisées pour exprimer les m contraintes du CSP.

Comme les clauses de domaines sont encodées en $o(nd)$ et les clauses de contraintes en $O(mad^a)$, la taille du codage *CGNF* pour un CSP est de $O(mad^a + nd)$ dans le pire des cas. En fait, on peut même considérer que c'est en $O(mad^a)$ étant donné que nd est souvent négligeable. Cette complexité spatiale est identique à celle du codage du CSP original, ce qui justifie l'optimalité de la complexité spatiale du codage *CGNF*. Les auteurs de (Bessière *et al.*, 2003) annoncent une complexité spatiale dans lepire des cas en $O(mad^a)$ pour le codage k-AC. D'après ce que nous avons compris de ce calcul, cette complexité est arrondie et ne tient compte ni des clauses "at-least-one", ni des clauses "at-most-one" de ce codage. Ce qui augmente sa complexité à $O(mad^a + nd^2)$.

Définition 1
Soit P un CSP et C son codage CGNF. Soit I une interprétation de C. On définit l'instantiation équivalente à I pour le CSP P l'instantiation I_p qui vérifie la condition suivante : pour toute variable CSP X et pour toute valeur v de son domaine, $X = v$ dans I_p si et seulement si $I[X_v]$ =vrai.

Théorème 1
Soient P un CSP, C son codage CGNF, I une interprétation du codage CGNF et I_p l'instantiation équivalente à I pour le CSP P. I est un modèle de C si et seulement si I_p est une solution de P.

Preuve 1
Soit I un modèle de C et I_p son instantiation équivalente dans P. I satisfait chaque clause de C, car c'est un de ses modèle. Nous devons prouver que chaque variable du CSP est assignée a une unique valeur de son domaine dans I_p et que I_p satisfait toutes les contraintes de P. Si une clause de domaine $C_{D_X} = (\pm 1, X_{v_1}\ X_{v_2} \ldots X_{v_d})$ de C est satisfaite, cela signifie qu'une et une seule variable booléenne parmi $\{X_{v_1}\ X_{v_2} \ldots X_{v_d}\}$ est vraie dans I. Comme toutes les clauses de domaine sont satisfaites, chaque variable du CSP est affectée une et une seule fois. De plus, chaque clause de contrainte $C_{C_i} = f_1 \vee f_2 \vee \ldots \vee f_m$ a au moins une de ses f_i satisfaite dans I. Ce qui signifie que les valeurs affectées au variables CSP impliquées dans la contrainte C_i associée à C_{C_i} forme un tuple de la relation R_i. Ainsi, C_i est satisfaite par I_p et toutes les contraintes de P sont satisfaites par I_p. Donc I_p est une solution de P.

La réciproque se démontre de manière analogue.

Le codage *CGNF* permet une représentation optimale des CSP, cependant il ne capture pas la propriété de consistance d'arc qui est la clé de presque tous les algorithmes énumératifs des CSP. Nous introduisons dans la section suivante une règle d'inférence simple qui s'applique sur le codage *CGNF* et prouvons que la consistance d'arc peut être établie en temps linéaire en appliquant la règle jusqu'à saturation.

4 Une nouvelle règle d'inférence pour le codage CGNF

Cette règle est basée sur la structure du CSP qui est codée par les clauses de domaine et de contrainte.

4.1 Définition de la règle d'inférence IR

Définition 2
Soient P un CSP, C son codage CGNF, C_{C_i} une clause de contrainte, C_{D_X} une clause de domaine, L_{C_i} l'ensemble des littéraux figurant dans C_{C_i} et L_{D_X} l'ensemble des littéraux de C_{D_X}. Si $L_{C_i} \cap L_{D_X} \neq \emptyset$ alors on infère chaque négation $\neg X_v$ d'un littéral positif[5] *qui figure dans L_{D_X} mais pas dans L_{C_i}. Nous obtenons la règle suivante :*
IR : si $L_{C_i} \cap L_{D_X} \neq \emptyset$, $X_v \in L_{D_X}$ et $X_v \notin L_{C_i}$ alors $C_{D_X} \wedge C_{C_i} \vdash \neg X_v$[6].

Exemple 1
Soit C_{D_B}=$(\pm 1, B_{v_0}\ B_{v_1})$ la clause de domaine correspondant à la variable B et C_{C_i}=$(A_{v_1} \wedge B_{v_0} \wedge C_{v_0}) \vee (A_{v_0} \wedge B_{v_0} \wedge C_{v_1})$ une clause de contrainte correspondant à la contrainte du CSP C_i impliquant les variables $\{A, B, C\}$. L'application de la règle IR sur ces deux clauses donne : $C_{D_B} \wedge C_{C_i} \vdash \neg B_{v_1}$.

Proposition 1
La règle d'inférence IR est correcte.

Preuve 2
Soient X_v un littéral figurant dans L_{D_X} mais pas dans L_{C_i} et I un modèle de $C_{C_i} \wedge C_{D_X}$. La clause de contrainte C_{C_i} est une disjonction de conjonctions f_i et chaque conjonction f_i contient un littéral de L_{D_X}. Au moins une des conjonctions f_i, disons f_j, est satisfaite par I car I est un modèle de C_{C_i}. Ainsi, il y a un littéral $X_{v'}$ ($X_{v'} \neq X_v$ car $X_v \notin L_{C_i}$) de f_j qui figure dans L_{D_X}, et tel que $I[X_{v'}]$ =vrai. À cause de l'exclusion mutuelle des littéraux de C_{D_X}, $X_{v'}$ est le seul littéral de L_{D_X} satisfait par I. Ainsi, $I[\neg X_v]$ =vrai et I est un modèle de $\neg X_v$.

4.2 La règle d'inférence et la consistance d'arc

Définition 3
Un CSP P est arc consistant si et seulement si tous ces domaines sont arc consistants. Un domaine $D_{X_{i_1}}$ est arc consistant si et seulement si pour chaque valeur v_{i_1} de $D_{X_{i_1}}$ et pour chaque contrainte C_j d'arité k impliquant les variables $\{X_{i_1}, \ldots, X_{i_k}\}$ il existe un tuple $(v_{i_2}, \ldots, v_{i_k}) \in D_{X_{i_2}} \times \ldots \times D_{X_{i_k}}$ tel que $(v_{i_1}, v_{i_2}, \ldots, v_{i_k}) \in R_j$.

Nous utilisons la règle d'inférence IR sur le codage CGNF C d'un CSP P, pour établir la consistance d'arc. Nous montrons qu'en appliquant IR sur C jusqu'à saturation (un point fixe) et en propageant les mono-littéraux négatifs inférés nous maintenons la consistance d'arc sur C avec une complexité linéaire.

[5] Dans le codage CGNF d'un CSP, il n'apparaît que des littéraux positifs.

[6] $\vdash$ représente l'inférence logique.

Proposition 2
Soient P un CSP et C son codage CGNF. Une valeur $v \in D_Y$ est filtrée par consistance d'arc dans P si et seulement si la négation $\neg Y_v$ de la variable booléenne correspondante est inférée par l'application de IR sur C.

Preuve 3
Soit v une valeur du domaine D_Y qui ne vérifie pas la propriété de consistance d'arc. Il y a au moins une contrainte C_j impliquant Y telle que v n'apparaisse dans aucun des tuples autorisés de la relation correspondante R_j. La variable booléenne Y_v n'apparaît pas dans la clause de contrainte associée C_{C_j}, mais elle apparaît dans la clause de domaine C_{D_Y} associée à D_Y. Si on applique la règle IR sur C_{D_Y} et C_{C_j} on infère $\neg Y_v$.

La réciproque se démontre de la même manière.

Théorème 2
Soient P un CSP et C son codage CGNF. La saturation de IR sur C et la propagation de tous les littéraux inférés est équivalent au filtrage par consistance d'arc dans le CSP P original.

Preuve 4
C'est une conséquence de la proposition 2.

4.3 Obtenir la consistance d'arc en appliquant IR

Pour programmer un filtrage par consistance d'arc sur C en appliquant IR, nous avons besoin de définir quelques structures de données. On suppose que les nd variables booléennes de C sont encodées par les premiers entiers $[1..nd]$. On définit un tableau OCC_j de taille nd pour chaque clause de contrainte C_{C_j} de telle sorte que $OCC_j[i]$ contient le nombre d'occurrences de la variable i dans C_{C_j}. Il y a un total de m tableaux de ce type qui correspondent aux m clauses de contrainte. Si $OCC_j[k] = 0$ pour un $k \in \{1 \dots nd\}$ et un $j \in \{1 \dots m\}$, la négation $\neg i$ de la variable booléenne i est inférée par IR. Cette structure de donnée ajoute un facteur $O(mnd)$ en complexité spatiale. La complexité spatiale totale de C est $O(mad^a + nd + mnd)$, mais les facteurs nd et mnd sont toujours inférieurs au facteur mad^a, donc la complexité spatiale de C reste en $O(mad^a)$.

Le principe du filtrage par consistance d'arc consiste tout d'abord à lire les m tableaux OCC_j pour détecter les variables $i \in [1 \dots nd]$ qui ont un nombre d'occurrences nul ($OCC_j[i] = 0$). Ceci est fait par les lignes 3 à 7 de l'Algorithme 1 en $O(mnd)$, car il y a m tableaux de taille nd. Ensuite, il faut appliquer de la propagation unitaire sur les variables détectées, et propager leur effet jusqu'à saturation (i.e. plus de nouvelle variable i ayant $OCC_j[i] = 0$). La procédure de consistance d'arc est décrite dans l'Algorithme 1. Elle fait appel à la procédure *Propager* décrite dans l'Algorithme 2.

La complexité de la procédure d'arc consistance est principalement donnée par l'effet de la propagation des littéraux de la liste L (ligne 8 à 11 de l'algorithme 1). Il est aisé de voir que dans le pire des cas il y aura nd appels à la procédure *propager*. Toutes les propagations dues à ces appels sont effectuées en $O(mad^a)$ dans le pire des cas. En

Algorithme 1 Consistance d'arc

Procédure Consistance d'arc(instance CGNF C)
1. **var** L : liste de littéraux
2. $L = \emptyset$
3. **pour** chaque clause de contrainte C_{C_j}
4. **pour** chaque littéral i de OCC_j
5. **si** $OCC_j[i] = 0$ **alors** ajouter $\neg i$ dans L **finsi**
6. **finpour**
7. **finpour**
8. **tant que** $L \neq \emptyset$ et $\square \notin C$
9. extraire $\neg l$ de L
10. propager (C, $\neg l$, L)
11. **fin tant que**

Algorithme 2 Propager

Procédure Propager (instance CGNF C, littéral $\neg i$, Liste L)
1. **si** i n'est pas encore affecté
2. **alors**
3. affecter i à la valeur *faux*
4. **pour** chaque conjonction non affectée f de C contenant i
5. affecter f à la valeur *faux*
6. **pour** chaque littéral j de f tel que $i \neq j$
7. soustraire 1 à $OCC_k[j]$
(k étant l'indice de la clause de contrainte contenant f)
8. **si** $OCC_k[j] = 0$ **alors** ajouter j dans L **finsi**
9. **finpour**
10. **finpour**
11. **finsi**

fait, il y a au plus d^a conjonctions de a littéraux pour chaque clause de contrainte de C. Le nombre total de conjonctions traitées à la ligne 5 de l'algorithme 2 ne peut pas excéder md^a car chaque conjonction f considérée à la ligne 4 est retirée à la ligne 5 (f est interprétée à faux). Comme il y a a littéraux par conjonction f, la propagation totale s'effectue en $O(mad^a)$. Ainsi, la complexité de la procédure de consistance d'arc est $O(mad^a + mnd)$. Mais une fois de plus, le facteur mnd est souvent négligeable devant mad^a. La complexité est donc réduite à $O(mad^a)$. Elle est linéaire dans la taille de C.

5 Deux méthodes énumératives pour le codage CGNF

Nous étudions dans cette partie deux méthodes énumératives : la première (MAC) maintient la consistance d'arc pendant la recherche tandis que la seconde (FC) ne main-

tient qu'une forme partielle de consistance d'arc, comme le fait la méthode du *forward checking* classique dans les CSP (Haralick & Elliott, 1980). Ces deux méthodes effectuent une énumération booléenne et des simplifications. Elle sont basées sur une adaptation de la procédure DPLL au codage *CGNF*.

5.1 La méthode MAC

MAC commence par un premier appel à l'Algorithme 1 pour vérifier la consistance d'arc à la racine de l'arbre de recherche. Ensuite, il se contentera de différents appels à la procédure *propager* décrite pas l'Algorithme 2 à chaque noeuds de l'arbre pour maintenir la consistance d'arc pendant le recherche. C'est à dire, les mono-littéraux de chaque noeuds sont inférés et leurs effets sont propagés. Le code de *MAC* est décrit dans l'Algorithme 3.

Algorithme 3 Algorithme *MAC* pour le codage CGNF.

Procédure MAC(instance CGNF C)
1. Consistance d'arc(C)
2. **si** $\square \in C$ **alors retourner**(insatisfaisable)
3. **sinon début**
4. choisir un littéral $l \in C$
5. **si** $Satisfaisable(C, l, vrai)$ **alors retourner**(satisfaisable)
6. **sinon si** satisfaisable$(C, l, faux)$ **alors retourner**(satisfaisable)
7. **sinon retourner**(insatisfaisable)
8. **fin**

5.2 La méthode FC

Il est aisé d'obtenir un algorithme équivalent à *Forward Checking (FC)* à partir de celui de MAC pour le codage CGNF. Le principe est le même que pour *MAC*, excepté qu'au lieu de maintenir la consistance d'arc sur C à chaque noeud de l'arbre de recherche, (FC) ne la maintient que sur le voisinage proche de la variable qui vient d'être instanciée. Ceci est fait en restreignant l'effet de la propagation de l'affectation du littéral aux seuls littéraux de la clause de domaine dans laquelle il apparaît. Il est important de noter qu'il est trivial de trouver une version de FC pour notre codage, alors que ce n'est pas le cas pour les CSP n-aires, où on trouve pas moins de six versions différentes de *FC* (Bessière *et al.*, 2002).

5.3 Heuristiques de choix de variable

Étant donné que le codage CGNF concerve la structure du CSP, on peut trouver aisément des heuristiques de choix de variables ou de valeurs équivalentes à toutes celles utilisées dans les CSP. Par exemple, l'heuristique du domaine minimum (MD) qui consiste à choisir, pendant la recherche, la variable CSP dont le domaine est le

Algorithme 4 Fonction Satisfaisable.

Fonction Satisfaisable(instance CGNF C, variable l, booléen val) :booléen
1. **var** L : liste de littéeraux
2. $L = \emptyset$
3. **si** $val = vrai$ **alors début**
4. affecter l à la valeur *vrai*
5. ajouter chaque littéral $i \neq l$ de la clause de domaine contenant l dans L
6. **tant que** $L \neq \emptyset$ et $\square \notin C$
7. extrairet i de L
8. propager (C, i, L)
9. **fin**
10. **sinon faire** $Propager(C, l, L)$ **tant que** $L \neq \emptyset$ et $\square \notin C$
11. **si** $C = \emptyset$ **alors retourner**(*vrai*)
12. **sinon si** $\square \in C$ **alors retourner**(*faux*)
13. **sinon début**
14. choisir un littéral p de C
15. **si** $satisfaisable(C, p, vrai)$ **alors retourner**(*vrai*)
16. **sinon si** $satisfaisable(C, p, faux)$ **alors retourner**(*vrai*)
17. **sinon retourner**(*faux*)
18. **fin**

plus petit, est équivalent dans le codage CGNF, à choisir un littéral qui apparaît dans la clause de domaine la plus courte. Cette heuristique est implémentée dans les méthodes MAC et FC.

6 Expérimentations

Nous avons expérimenté les deux méthodes *MAC* et *FC* et comparé leurs performances sur des instances du problème de Ramsey et sur des CSP à contraintes d'arités 3 et 4 générés aléatoirement encodés sous formes CGNF. Ces programmes sont écrits en C de manière itérative , compilés et exécutés sous Windows avec un processeur 2.8Gh et 1Go de RAM.

6.1 Le problème de Ramsey

Le problème de Ramsey (voir CSPlib) consiste à colorer les arêtes d'un graphe complet à n sommets à l'aide de k couleurs de telle sorte qu'il n'y ai pas de triangle monochromatique. Le Tableau 1 montre les résultats de *MAC* et de *FC* sur quelques instances du problème de Ramsey où $k = 3$ et n varie entre 5 et 14. Ces problèmes ont des solutions. La première colonne definie le problème : Rn_k dénote l'instance du problème de Ramsey avec n sommets et k couleurs, la seconde et la troisième colonnes montrent les nombres de noeuds et les performances en temps CPU de *FC* et respectivement *MAC* augmentés par l'heuristique (MD) décrite précédemment.

Problème	FC + MD		MAC + MD	
	Noeuds	Temps	Noeuds	Temps
R5_3	*12*	*0 s 48 μs*	*12*	*0 s 85 μs*
R6_3	*27*	*0 s 231 μs*	*21*	*0 s 297 μs*
R11_3	*237*	*0 s 5039 μs*	*103*	*0 s 4422 μs*
R12_3	*538*	*0 s 21558 μs*	*130*	*0 s 11564 μs*
R13_3	*1210*	*0 s 28623 μs*	*164*	*0 s 9698 μs*
R14_3	*216491*	*9 s 872612 μs*	*6735*	*1 s 752239 μs*

FIG. 1 – Résultats de *MAC* et *FC* pour quelques problèmes de Ramsey.

On peut voir que *FC* est meilleur que *MAC* pour les instances de petite taille ($n \leq 6$) mais visite plus de noeuds. Cependant, *MAC* est meilleur que *FC* en temps et en nombre de noeuds visités dès lors que la taille du problème augmente ($n \geq 7$). *MAC* surpasse *FC* dans la plupart des cas pour ce problème.

6.2 Les problèmes aléatoires

La seconde classe de problème étudiée correspond aux CSP générés aléatoirement. Nous avons mené des expériences sur des CSP à contraintes d'arité 3 (3-CSP) et 4 (4-CSP) générés aléatoirement. Dans les deux cas, le nombre de variables est 10. Le nombre de valeurs par domaine est de 10 pour les 3-CSP et de 5 pour les 4-CSP. Notre générateur est une adaptation pour la forme CGNF du générateur de Bessière et al.. Il utilise 5 paramètres : n le nombre de variables, d la taille des domaines, a l'arité des contraintes, $dens$ la densité des contraintes qui est le rapport entre le nombre de contraintes et le nombre maximum de contraintes possibles, et t la dureté qui est la proportion de tuples interdits de chaque contraintes. Les CSP obtenus ainsi sont exprimés sous forme CGNF.

La Figure 2 (respectivement Figure 3) montre les courbes moyennes représentant les temps CPU pour MAC et FC augmentés par l'heuristique MD par rapport à une variation de la dureté des contraintes sur les 3-CSP (respectivement 4-CSP). Les deux courbes de droite de la Figure 2 (respectivement Figure 3) sont celles de MAC et FC pour une faible densité : *dens*=0,20 (courbes A) (respectivement *dens*=0,096 (courbe D)), les deux au milieu correspondent à une densité moyenne : *dens*=0,50 (courbes B) (respectivement *dens*=0,50 (courbes E)) et les deux de gauches correspondent à une densité haute : *dens*=0,83 (courbes C) (respectivement *dens*=0,96 (courbes F)). On peut voir que le pic de difficulté de la région difficile pour chaque classe de problème correspond à une valeur critique de dureté et que ce pic de difficulté augmente si la densité augmente. Plus la densité est faible, plus la dureté critique correspondante est grande. On peut également noter que *MAC* surpasse définitivement *FC* sur les six classes de problèmes.

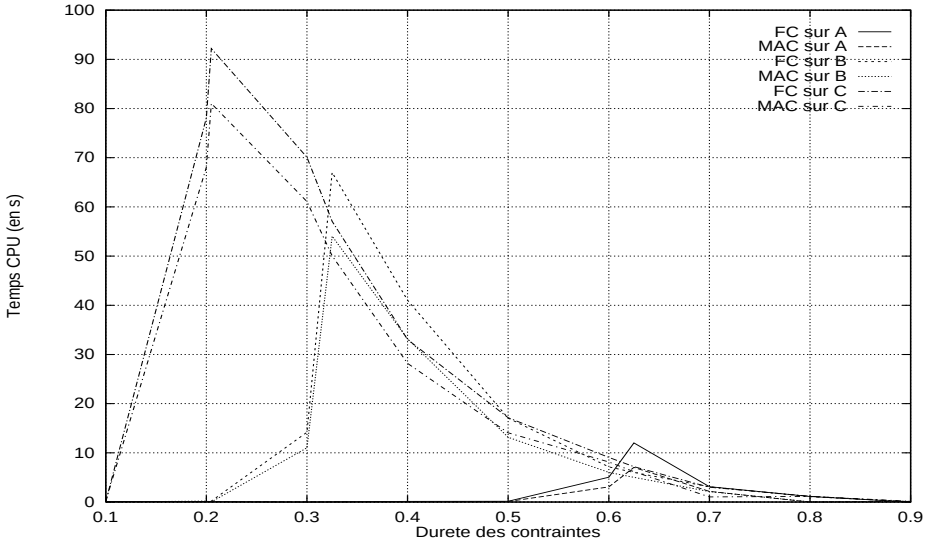


FIG. 2 – Résultats de MAC et FC sur des 3-CSP ayant pour densités : dens=0,20, dens=0,50 et dens=0,83.

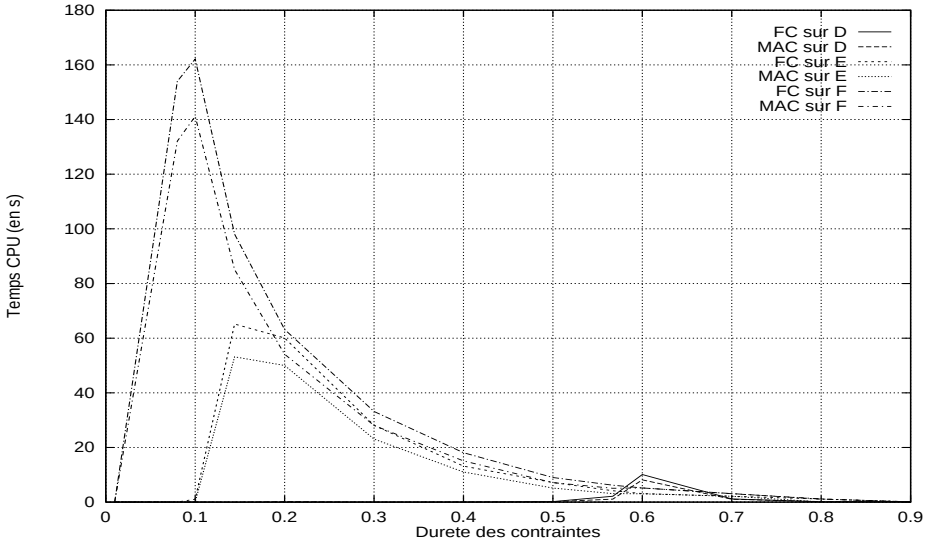


FIG. 3 – Résultats de MAC et FC sur des 4-CSP ayant pour densités : dens=0,096, dens=0,50 et dens=0,95.

7 Travaux de référence, discussion et comparaison

Nous résumons ici quelques transformations bien connues de CSP vers SAT tout en comparant leurs propriétés.

- *Le codage direct* (Kleer, 1989) est la transformation de CSP vers SAT la plus utilisée. Pour un CSP P, le codage direct a une complexité en espace dans le pire des cas en $O(nd + \frac{nd(d-1)}{2} + mad^a)$ et ne capture pas la consistance d'arc.
- *Le codage AC* (Kasif, 1990) est défini pour les CSP binaires. Sa particularité est de représenter la propriété de consistance d'arc dans le codage. Il permet à une procédure simple SAT (la propagation unitaire) d'obtenir la consistance d'arc. Sa complexité en espace dans le pire des cas est la même que celle du codage direct.
- *Le codage k-AC* (Bessière *et al.*, 2003) est une généralisation du codage AC au CSP n-aires. Sa complexité spatiale est identique à celle des deux autres, et légèrement supérieure à celle du codage CGNF qui est en $O(nd + mad^a)$. Ce codage capture la *k-consistance d'arc relationnelle* (Bessière *et al.*, 2003) et la (i, j)*-consistance* (Freuder, 1985), mais le nombre de clauses nécessaire au maintient du codage sous la forme CNF, et les extra variables introduites pour exprimer les supports peuvent, dans certains cas, ralentir un solveur SAT.
- *Comparaison et discussion* : Notre approche est différente de ces travaux précédents. Elle consiste en une généralisation de la forme CNF qui permette un codage naturel et optimal de CSP n-aires, tout en gardant leur structure. Ainsi, on peut profiter des avantages de SAT et des CSP. Notre but dans ce papier est d'introduire une nouvelle méthode prometteuse et de montrer sa praticabilité. Nous ne cherchons pas, à ce niveau, à optimiser nos codes et heuristiques pour rivaliser avec les plus récentes implémentations des algorithmes les plus efficaces. Nous proposons juste une première implémentation qui se comporte sensiblement comme les nFC pour CSP n-aires (Bessière *et al.*, 2002). Nous pensons qu'un solveur SAT sophistiqué comme *Chaff* (Malik *et al.*, 2001) aurait de meilleurs performences que notre implémentation, si on l'appliquait sur le codage $k-$AC de CSP n-aires. Mais il est évident que toutes les améliorations apportées à la procédure DPLL qui ont menée à *Chaff* peuvent être adaptées aisément à notre méthode. De telles optimisations augmenteraient les performences de nos algorithmes de manière notable, cette question sera traitée dans un futur papier.

8 Conclusions

Nous avons étudié une généralisation de la représentation *CNF* qui permet un codage booléen efficace pour les CSP n-aires. Nous avons montré que la taille du codage booléen d'un CSP est identique à celle qu'il a dans son formalisme original. Nous avons aussi prouvé que la consistance d'arc est effectuée en temps linéaire avec ce nouveau formalisme. Nous avons implémenté une nouvelle règle d'inférence dont l'application jusqu'à saturation établie la consistance d'arc pour les CSP n-aires exprimés dans le codage booléen. Du fait que l'application de cette règle est basée sur la propagation unitaire, son efficacité n'est pas sensible à l'arité des contraintes du CSP. Deux méthodes énumératives sont proposées : la première (MAC), maintient la consistance d'arc à chaque noeud de l'arbre de recherche tandis que la seconde (FC) ne maintient qu'une forme partielle de consistance d'arc. Ces deux méthodes sont bien connues dans les CSP et se récupèrent aisément dans notre codage booléen. Elles ont été expérimentées

sur des instances du problème de Ramsey ainsi que sur des CSP à contraintes d'arité 3/4 générés aléatoirement et nous avons obtenu des résultats qui montrent que le maintient de la consistance d'arc complète dans le codage booléen est la meilleure solution.

Ce travail pourrait être étendu de différente manières. Il serait intéressant de trouver une règle d'inférence qui permettrait de récupérer la consistance de chemin dans le codage booléen. Un autre point serait de regarder les classes polynomiales de ce codage. Et enfin, détecter et supprimer les symétries dans ce codage augmenterait l'efficacité des méthodes énumératives que nous proposons.

Références

BENNACEUR H. (1996). The satisfiability problem regarded as constraint satisfaction problem. In *Proceedings of ECAI'96*, p. 155–159.

BESSIÈRE C., HEBRARD E. & WALSH T. (2003). Local consistency in sat. In *International Conference on Theory and Application of satisfiability testing*, p. 400–407.

BESSIÈRE C., MESEGUER P., FREUDER E. C. & J.LARROSA (2002). On forwardchecking for non-binary constraint satisfaction. *Journal of Artificial Intelligence*, **141**, 205–224.

D. SABIN E. F. (1997). Understanding and improving the mac algorithm. In *Proceeding of CP'97*, p. 167–181.

DAVIS M., LOGEMANN G. & LOVELAND G. (1962). A machine program for theorem proving. *Comm. ACM 5*, p. 394–397.

DAVIS M. & PUTNAM H. (1960). A computing procedure for quantification theory. *Journal of ACM 7*, p. 201–215.

FREUDER E. C. (1985). A sufficient condition for backtrack-bounded search. *J. ACM*, **32**(4), 755–761.

GOLDBERG E. & NOVIKOV Y. (2002). Berkmin : A fast and robust sat solver. In *Proceedings of the 2002 Design Automation and Test in Europe*, p. 142–149.

HARALICK R. & ELLIOTT G. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, **14**, 263–313.

KASIF S. (1990). On the parallel complexity of descrete relaxation in constraint satisafaction networks. *Journal of Artificial Intelligence*, **45**, 275–286.

KLEER J. D. (1989). A comparison of atms and csp techniques. In *Proceedings of IJCAI'89*, p. 290–296, DETROIT.

MALIK S., ZHAO Y., MADIGAN C. F., ZHANG L. & MOSKEWICZ M. W. (2001). Chaff : Engineering an efficient sat solver. In *IEEE 2001*, p. 530–535, Las Vegas, United States.

MONTANARI U. (1974). Networks of constraints : Fundamental properties and application to picture processing. *Journal Inform. Sci.*, **9**(2), 95–132.

ROSSI F., PETRI C. & DHAR V. (1990). On the equivalence of constraint satisfaction problems. In *Proceedings of ECAI-90*, p. 550–556.

RUSSEL O. (2004). An other sat to csp conversion. p. 558–565.

SILVA J. & SAKALLAH K. (1996). Grasp – a new search algorithm for satisfiability. In *IEEE 1996*, p. 220–227, San Jose, California, United States.

WALSH T. (2000). Sat v csp. In *Proceedings of CP'00*, p. 441–456.

Une heuristique de branchement dirigée vers les formules Horn renommables quantifiées

Florian Letombe

Centre de Recherche en Informatique de Lens, CNRS FRE 2499
Faculté des Sciences Jean Perrin
Université d'Artois
F-62307, Lens Cedex, France
letombe@cril.univ-artois.fr

Résumé : Ces dernières années, le problème de validité des Formules Booléennes Quantifiées ($\mathcal{QBF}$s) est apparu comme un sujet important de l'IA ; de nombreux prouveurs ont été construits dans le but de le résoudre. Dans ce papier, nous décrivons une nouvelle heuristique de branchement pour les prouveurs QBF basés sur l'architecture DPLL. Cette heuristique vise à promouvoir des formules Horn renommables après simplification. Cette heuristique est basée sur l'algorithme de reconnaissance de formules Horn renommables proposé par Hébrard (Hébrard, 1994). Nous présentons des résultats expérimentaux obtenus par notre prouveur QBF **Qbfl** avec la nouvelle heuristique de branchement et montrons dans quelle mesure ses performances sont améliorées.

Mots-clés : Formules Booléennes Quantifiées, Logique propositionnelle, classes polynomiales.

1 Introduction

QBF, le problème de validité pour les $\mathcal{QBF}$s a une importance croissante en IA. Cela peut s'expliquer par le fait que, en tant que problème PSPACE-complet canonique, de nombreux problèmes d'IA peuvent être réduits à QBF de manière polynomiale (cf. (Egly *et al.*, 2000; Fargier *et al.*, 2000; Besnard *et al.*, 2002; Rintanen, 1999a; Pan *et al.*, 2002; Pan & Vardi, 2003)) ; de plus, pour différents domaines de l'IA (parmi lesquels la planification, le raisonnement non monotone, l'inférence paraconsistante), une approche basée sur une traduction vers $\mathcal{QBF}$ peut se révéler plus « efficace » que des algorithmes dépendants du domaine dédiés. Par conséquent, de nombreux prouveurs QBF ont été conçus (cf. principalement (Cadoli *et al.*, 1998; Rintanen, 1999b; Feldmann *et al.*, 2000; Rintanen, 2001; Giunchiglia *et al.*, 2001a; Letz, 2002; Zhang & Malik, 2002; Benedetti, 2004; GhasemZadeh *et al.*, 2004; Pan & Vardi, 2004; Audemard & Saïs, 2004)) et évalués ces dernières années (Le Berre *et al.*, 2003, 2005).

QBF est un problème calculatoirement difficile, aussi bien en théorie qu'en pratique. Les méthodes basées sur la notion de restriction comptent parmi celles permettant

d'augmenter l'ensemble d'instances traitables en pratique. L'idée clé est de reconnaître les instances pour lesquelles des algorithmes spécifiques peuvent fournir une solution beaucoup plus efficacement que les prouveurs QBF généraux. Plusieurs restrictions traitables de $\mathcal{QBF}$ ont déjà été identifiées. (Aspvall *et al.*, 1979; Gent & Rowley, 2002) montrent que les formules de Krom quantifiées ($\mathcal{QKF}$s) sont décidables en temps polynomial. (Kleine-Büning *et al.*, 1995) ont prouvé que les formules de Horn quantifiées ($\mathcal{QHF}$s) forment une restriction traitable de $\mathcal{QBF}$. Par conséquent, les formules Horn renommables quantifiées (ren$\mathcal{QHF}$s) sont également décidables de façon polynomiale.

Le principal objectif de cet article est de présenter une nouvelle heuristique de branchement pour les prouveurs QBF basés sur DPLL[1], une méthode énumérative à la base de la plupart des algorithmes complets pour SAT, le problème bien connu de la satisfaisabilité d'une formule propositionnelle sous forme normale conjonctive (qui constitue une restriction de QBF. Cette nouvelle heuristique a pour but de favoriser la génération de ren$\mathcal{QHF}$s, pour lesquelles des prouveurs efficaces existent.

La suite de cet article est organisée comme suit. Quelques préliminaires formels sont donnés au paragraphe 2. La nouvelle heuristique de branchement est présentée au paragraphe 3. Le paragraphe 4 décrit notre prouveur **Qbfl**. Les résultats expérimentaux obtenus par Qbfl avec cette heuristique sont présentés au paragraphe 5. Enfin, le paragraphe 6 conclut le papier et donne quelques perspectives.

2 Formules Booléennes Quantifiées

On désigne par $PROP_{PS}$ le langage propositionnel construit à partir d'un ensemble fini PS de symboles, les connecteurs usuels et les constantes booléennes $vrai$, $faux$ de manière standard.

2.1 $\mathcal{QBF}$

Définition 1 (Syntaxe d'une Formule Booléenne Quantifiée)
L'ensemble $QPROP_{PS}$ des formules booléennes quantifiées ($\mathcal{QBF}$s) sur PS est le plus petit ensemble de mots défini inductivement comme suit : [2]

1. $vrai$, $faux$ *et toute variable de* PS *appartient à* $QPROP_{PS}$.
2. *Si* ϕ *et* ψ *appartiennent à* $QPROP_{PS}$, *alors* $\neg(\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \Rightarrow \psi)$, $(\phi \Leftrightarrow \psi)$, $(\phi \oplus \psi)$ *appartiennent à* $QPROP_{PS}$.
3. *si* ϕ *appartient à* $QPROP_{PS}$ *et* x *appartient à* PS, *alors* $\forall x(\phi)$ *et* $\exists x(\phi)$ *appartiennent à* $QPROP_{PS}$.

Les occurrences de variables propositionnelles x dans une formule Σ de $QPROP_{PS}$ peuvent être partitionnées en trois ensembles : les occurrences *quantifiés*, *liées* et *libres* de x. Les occurrences quantifiées sont celles apparaissant dans une quantification, i.e., juste après un quantificateur $\forall$ ou $\exists$. Dans toute sous-formule $\forall x(\phi)$ (resp. $\exists x(\phi)$) de

[1] Du nom des auteurs de la technique : M. Davis, G. Logemann et D. Loveland (Davis *et al.*, 1962). Cette dernière a été initiée par M. Davis et H. Putnam (Davis & Putnam, 1960).

[2] Afin de simplifier la syntaxe, certaines parenthèses sont omises si cela n'affecte pas l'équivalence.

Σ, toutes les occurrences de x dans ϕ sont *liées*. Les occurrences de x sont dites *dans la portée de la quantification* $\forall x$ (resp. $\exists x$). Enfin, toutes les occurrences restantes de x dans Σ sont libres.

$Var(\Sigma)$ est l'ensemble des variables apparaissant dans Σ. Une variable x de Σ est *libre* si et seulement si x a une occurrence libre dans Σ.

Un *littéral* est une occurrence positive ou négative d'une variable.

Une formule Σ est dite *polie* si et seulement si chaque occurrence liée d'une variable x de Σ est dans la portée d'un quantificateur unique, et chaque variable libre n'a pas d'occurrence liée. Une formule Σ est dite sous forme *prénexe* si et seulement si $\Sigma = Qx_1(\ldots Qx_n(\phi)\ldots)$ où chaque occurrence de Q vaut soit $\forall$, soit $\exists$, et ϕ ne contient pas d'occurrence quantifiée de variable. ϕ est la *matrice* de Σ et la suite $Qx_1 \ldots Qx_n$ de quantifications est le *préfixe* de Σ. Une formule est dite *fermée* si et seulement si elle ne contient pas de variable libre. Une formule est dite *sans quantification* si et seulement si elle ne contient pas de quantification (de telles formules peuvent être considérées comme des formules propositionnelles « standard »). Le sous-ensemble de $QPROP_{PS}$ ne contenant que des formules sans quantification est noté $PROP_{PS}$. Notons parmi ces sous-ensembles le fragment des $\mathcal{CNF}$s (Formes Normales Conjonctives) contenant les conjonctions (finies) de clauses. Une formule en $\mathcal{CNF}$ est souvent vue comme un ensemble de clauses.

Pour chaque formule booléenne quantifiée Σ et chaque variable x, $\Sigma_{x\leftarrow 0}$ (resp. $\Sigma_{x\leftarrow 1}$) désigne la formule obtenue en remplaçant chaque occurrence libre de x de Σ par $faux$ (resp. $vrai$).

Puisqu'il est possible de permuter deux quantifications successives de la même nature dans une formule sans effet sur l'équivalence, pour tout sous-ensemble fini, non vide $S = \{x_1, \ldots, x_n\}$ de PS, notons $\forall S.(\phi)$ (resp. $\exists S.(\phi)$) pour désigner plus simplement $\forall x_1(\ldots \forall x_n(\phi)\ldots)$ (resp. $\exists x_1(\ldots \exists x_n(\phi)\ldots)$).

Toute $\mathcal{QBF}$ peut être transformée en temps polynomial en une formule équivalente sous forme prénexe et polie (les variables liées peuvent être renommées sans effet sur l'équivalence) dont la matrice est en $\mathcal{CNF}$. Pour la suite, les matrices des formules $\mathcal{QBF}$ considérées sont en $\mathcal{CNF}$. Cela n'a pas d'incidence sur la complexité du problème si le quantificateur le plus interne est existentiel. Le problème QBF est souvent défini comme suit.

Définition 2 (QBF)
QBF *est le problème de décision suivant :*
- **Entrée :** *Une formule sous forme prénexe, polie, fermée* Σ *de* $QPROP_{PS}$ *;*
- **Question :** Σ *est-elle valide ?*

2.2 Restrictions traitables de $\mathcal{QBF}$

Nous nous focalisons dans ce papier sur deux restrictions de $\mathcal{QBF}$s : $\mathcal{QHF}$s et ren$\mathcal{QHF}$s.

Définition 3 (Formule de Horn Quantifiée)
Une formule $\mathcal{QBF}$ *prénexe est* $\mathcal{QHF}$ *ssi sa matrice est une formule* $\mathcal{CNF}$ *dans laquelle chaque clause contient au plus un littéral positif.*

Notre heuristique de branchement orientée pour la génération de ren$\mathcal{QHF}$s est fondée sur l'algorithme de reconnaissance de formules Horn renommables proposé par Hébrard (Hébrard, 1994).

Avant de la présenter, nous devons rappeler quelques notions introduites par Hébrard (Hébrard, 1994). Rappelons déjà qu'un littéral est soit une variable $x \in PS$, soit une variable niée $\neg x$. Le littéral complémentaire de x (resp. $\neg x$) est $\neg x$ (resp . x). Soient Σ une $\mathcal{QBF}$ et L un ensemble de littéraux. L est *consistant* s'il ne contient pas p et $\neg p$, pour toute variable propositionnelle p. L est *complet* si p appartient à L ou $\neg p$ appartient à L, pour tout p de $Var(\Sigma)$. Un *renommage* R est un ensemble complet et consistant de littéraux.

R est un renommage Horn pour Σ si la formule est obtenue en remplaçant dans la matrice de Σ toute occurrence d'un littéral par son littéral complémentaire si le littéral négatif appartient à R est $\mathcal{QHF}$.

Définition 4 (Formule Horn Renommable Quantifiée)
Une formule $\mathcal{QBF}$ est dite ren$\mathcal{QHF}$ s'il existe un renommage Horn pour cette formule.

Définition 5 ($\Rightarrow$, $\Rightarrow^*$ et CLOS(l))
Soient l et t deux littéraux et Σ une $\mathcal{CNF}$ matrice d'une $\mathcal{QBF}$:
- *$l \Rightarrow t$ ssi il existe une clause $C \in \Sigma$ telle que $l \in C$, $\neg t \in C$ et $l \neq \neg t$.*
- *La fermeture réflexive transitive de $\Rightarrow$ est désignée par $\Rightarrow^*$.*
- *CLOS(l) désigne l'ensemble $\{t \mid l \Rightarrow^* t\}$.*

Exemple 1 ($\Rightarrow$, $\Rightarrow^*$)
Soit la matrice suivante :

$$(a \vee b \vee c) \wedge (a \vee \neg d \vee e) \wedge (b \vee \neg c \vee \neg d) \wedge (\neg c \vee \neg e) \wedge (\neg b \vee c)$$

$a \Rightarrow \neg b$ et $a \Rightarrow \neg c$ dans la première clause ; $\neg c \Rightarrow d$ dans la troisième clause ; $a \Rightarrow^* d$ car $a \Rightarrow \neg c$ et $\neg c \Rightarrow d$. Nous illustrons CLOS(l) au paragraphe 3.

Un ensemble de littéraux L est dit *fermé* si CLOS(l) $\subseteq L$, pour tout $l \in L$.

Proposition 1 (Proposition 1.1 de (Hébrard, 1994))
Un renommage R est Horn si et seulement s'il est fermé, i.e. pour tout $l \in R$, CLOS$(l) \subseteq R$.

Une esquisse de l'algorithme de renommage de variables d'Hébrard est présentée dans l'algorithme 1 (à gauche).

3 Une nouvelle heuristique de branchement

L'idée de notre heuristique est de brancher en priorité sur des variables pour lesquelles la propagation conduit à une formule Horn renommable, ou à une formule « presque » Horn renommable. Nous utilisons l'algorithme proposé par Hébrard pour détecter les formules Horn renommables. Cet algorithme construit un ensemble de littéraux à renommer afin d'obtenir une formule Horn. Si un tel ensemble existe, il est possible

Algorithme 1 : Algorithme de Horn renommage de Hébrard (à gauche) et calcul de la distance δ (à droite)

fonction RHCLOS
Données : une $\mathcal{QBF}$ **Q**
Résultat : $\emptyset$ si **Q** n'est pas Horn renommable,
un Horn renommage R pour **Q** sinon.
début
  $R \leftarrow \emptyset$;
  pour chaque *variable* **p** *de* **Var(Q) faire**
    si $p \notin R$ **et** $\neg p \notin R$ **alors**
      si $\forall q \in$ CLOS*(p)*$\backslash R$, $\neg q \notin$ CLOS*(p)*$\backslash R$ **alors**
        # CLOS$(p)\backslash R$ est consistant
        $R \leftarrow R \cup$ (CLOS$(p)\backslash R$) ;
      sinon si $\forall q \in$ CLOS*(¬p)*$\backslash R$, $\neg q \notin$ CLOS*(¬p)*$\backslash R$ **alors**
        # CLOS$(\neg p)\backslash R$ est consistant
        $R \leftarrow R \cup$ (CLOS$(\neg p)\backslash R$) ;
      sinon retourner $\emptyset$;
fin

fonction δ
Données : une $\mathcal{QBF}$ **Q**, un littéral l
Résultat : la distance δ de l à un Horn renommage.
début
  $Closl \leftarrow \{l\}$;
  $Distance \leftarrow 0$;
  ENFILER$(Filel, l)$;
  tant que non FILEVIDE*(Filel)* **faire**
    $m \leftarrow$ DEFILER$(Filel)$;
    $Engendre \leftarrow \{\neg t | m \Rightarrow t\}$;
    pour chaque $e \in Engendre$ **faire**
      si $\neg e \in Closl$ **alors**
        retourner $Distance + 1$;
      sinon si $e \in Closl$ **alors**
        $Closl \leftarrow Closl \cup \{e\}$;
        ENFILER$(Filel, e)$;
    $Distance \leftarrow Distance + 1$;
  retourner $Distance + 1$;
fin

d'utiliser l'algorithme polynomial de Kleïne-Büning (Kleine-Büning *et al.*, 1995) afin de résoudre l'instance. Sinon, nous utilisons l'algorithme pour mesurer à quelle distance nous sommes d'une ren$\mathcal{QHF}$: plus cette mesure est grande, plus nous sommes près d'une ren$\mathcal{QHF}$. Nous appelons cette mesure la *distance de contradiction*.

Définition 6 (Distance de contradiction)
La « distance »[3] *d'une contradiction de Horn renommabilité δ_l pour un littéral l dans une $\mathcal{QBF}$ Q est définie comme suit :*

$$\delta_l = \begin{cases} 0 \; si \nexists v | l \Rightarrow v \\ 1 \; si \; l \Rightarrow t \; et \; l \Rightarrow \neg t \\ 1 + min(\{\delta_v | l \Rightarrow v\}) \; sinon. \end{cases}$$

L'idée derrière cette distance est que, plus nous nous approchons d'une formule Horn renommable, plus la distance de contradiction est grande.

Afin de calculer cette distance, il est nécessaire de légèrement modifier l'algorithme de Hébrard : l'algorithme de Hébrard réalise une recherche en profondeur d'abord (DFS) alors que le nôtre utilise la recherche en largeur d'abord (BFS). C'est nécessaire si l'on veut trouver une distance minimale. La partie gauche de l'algorithme 1 décrit un tel calcul. Il est de complexité linéaire ($O(n + m)$, avec n le nombre de littéraux et m le nombre de clauses, si l'on considère que les clauses ont une longueur bornée).

[3]On notera l'abus de langage. Ce n'est pas une distance au sens mathématique.

Exemple 2 (Mesure en utilisant une DFS)
Considérons à nouveau la matrice de l'exemple 1. L'algorithme original réalise une DFS pour calculer la fermeture réflexive transitive de a :

- *Profondeur 0 :* $\textsc{Clos}(a) = \{a\}$ *;*
- *Profondeur 1 :* a *est dans les clauses* $(a \vee b \vee c)$ *et* $(a \vee \neg d \vee e)$ *;*
- $(a \vee \neg d \vee e) \rightsquigarrow \textsc{Clos}(a) = \{a, d, \neg e\}$ *;*
- *Profondeur 2 :* $\neg e$ *est dans la clause* $(\neg c \vee \neg e)$ *;*
- $(\neg c \vee \neg e) \rightsquigarrow \textsc{Clos}(a) = \{a, d, \neg e, c\}$ *;*
- *Profondeur 3 :* c *est dans les clauses* $(a \vee b \vee c)$ *et* $(\neg b \vee c)$ *;*
- $(a \vee b \vee c) \rightsquigarrow \textsc{Clos}(a) = \{a, d, \neg e, c, \neg a, \neg b\}$ *;*
- $\{a, d, \neg e, c, \neg a, \neg b\} \equiv \bot$.

Malheureusement, avec la DFS, aucune garantie ne nous permet de conclure que la contradiction trouvée est obtenue après un nombre minimal d'étapes.

Exemple 3 (Distance de contradiction (en utilisant une BFS))
L'algorithme modifié en BFS produira sur le même exemple le résultat suivant :

- *Niveau* $\delta_a = 0$ *:* $\textsc{Clos}(a) = \{a\}$ *;*
- *Niveau* $\delta_a = 1$ *:*
  - a *apparaît dans les clauses* $(a \vee b \vee c)$ *et* $(a \vee \neg d \vee e)$ *;*
  - $(a \vee b \vee c) \rightsquigarrow \textsc{Clos}(a) = \{a, \neg b, \neg c\}$ *;*
  - $(a \vee \neg d \vee e) \rightsquigarrow \textsc{Clos}(a) = \{a, \neg b, \neg c, d, \neg e\}$ *;*
- *Niveau* $\delta_a = 2$ *:*
  - $\neg b$ *apparaît dans la clause* $(\neg b \vee c)$ *;*
  - $\neg c$ *apparaît dans les clauses* $(b \vee \neg c \vee \neg d)$ *et* $(\neg c \vee \neg e)$ *;*
  - $(\neg c \vee \neg e) \rightsquigarrow \textsc{Clos}(a) = \{a, \neg b, \neg c, d, \neg e, e\}$ *;*
  - $\{a, \neg b, \neg c, d, \neg e, e\} \equiv \bot$.

La contradiction apparaît au second niveau. Nous concluons ainsi que a est à une distance de contradiction de 2 d'une formule Horn renommable.

Cette distance est souvent insuffisante pour élire une seule variable. C'est pourquoi nous utilisons cette distance dans une heuristique à la Jeroslow-Wang (Jeroslow & Wang, 1990) avec un réglage largement utilisé dans les prouveurs complets pour des k-SAT aléatoires hérités de POSIT (Freemann, 1995). Nous restreignons simplement la sélection de variables dans la portée du quantificateur le plus externe et nous branchons en priorité sur le littéral obtenant la plus grande distance de contradiction.

Définition 7 (Heuristique Δ de Horn Renommabilité)
Soit X_1 le groupe de quantificateurs le plus externe. Notre heuristique Δ est comme suit :

- $\Delta_x = 1024 \times \delta_x \times \delta_{\neg x} + \delta_x + \delta_{\neg x}$ *;*
- *la variable x est choisie si $x \in X_1$ et, $\forall y \in X_1$, $(x \neq y \Rightarrow \Delta_x \leq \Delta_y)$;*
- *le littéral x est enfin choisi si $\delta_x > \delta_{\neg x}$, $\neg x$ sinon.*

4 Qbfl

Qbfl[4] est un prouveur QBF qui utilise le prouveur SAT Limmat (version 1.3) comme un oracle SAT. Il est basé sur un procédure DPLL standard avec un branchement et/ou dépendant du groupe de quantificateurs (Cadoli *et al.*, 2002), fausseté et vérité triviale (Cadoli *et al.*, 2002). Il est adaptatif dans le sens où la fréquence des tests de trivialité évolue avec son taux de succès. La propagation de littéraux unitaires et purs pour $\mathcal{QBF}$ (Cadoli *et al.*, 2002) est également implantée. **Qbfl** réalise une détection de ren$\mathcal{QHF}$s et les résout à l'aide de l'algorithme proposé par Kleine-Büning *et al.* (Kleine-Büning *et al.*, 1995).

Plus précisément, l'algorithme implanté dans notre prouveur est basé sur celui proposé dans (Giunchiglia *et al.*, 2001b). Il commence par simplifier la formule, i.e. réalise la propagation standard des littéraux unitaires et purs. Il teste ensuite si une vérité ou une fausseté triviale est atteinte. Si ce n'est pas le cas, il détecte les ren$\mathcal{QHF}$s et les résout le cas échéant.

Qbfl résout les ren$\mathcal{QHF}$s à l'aide de la Q-Unit-Resolution de Kleine-Büning *et al.* (Kleine-Büning *et al.*, 1995) munie du renommage de variables.

Qbfl contient plusieurs heuristiques parmi lesquelles celle décrite dans ce papier et une adaptation aux $\mathcal{QBF}$s de celle proposée par Jeroslow et Wang (Jeroslow & Wang, 1990). Cette dernière est celle qui a fonctionné le mieux lors de l'évaluation $\mathcal{QBF}$ de 2004 (Le Berre *et al.*, 2005).

Ces deux heuristiques sont au cœur de nos expérimentations.

5 Résultats expérimentaux

5.1 Méthodologie

Les résultats empiriques présentés dans cette section ont été obtenus sur des PIV 3GHz et de 512MB de RAM.

Afin de comparer Qbfl (version 1.7) avec d'autres prouveurs, nous avons choisi d'utiliser QuBE-Rel[5] (version 1.3) et Semprop[6] (version 010604) pour ces expérimentations : ces deux prouveurs se sont révélés parmi les meilleurs des prouveurs QBF ayant participé aux deux évaluations $\mathcal{QBF}$ (Le Berre *et al.*, 2003, 2005) qui sont librement accessibles.

Dans les tableaux suivants, nous prenons en compte uniquement les temps CPU des instances résolues. Cependant, il est également nécessaire de vérifier le nombre de benchmarks résolus afin de comparer les comportements des prouveurs.

[4] `http://www.cril.univ-artois.fr/~letombe/qbfl`

[5] `http://www.star.dist.unige.it/~qube`

[6] `http://www4.in.tum.de/~letz/semprop`

5.2 Benchmarks « polynomiaux »

Nos premières expérimentations concernent des ensembles de benchmarks de $\mathcal{QHF}$s et ren$\mathcal{QHF}$s engendrés aléatoirement[7]. Les instances sont engendrées comme suit.

$\mathcal{QHF}$ Pour chaque clause, nous choisissons aléatoirement sa taille. Ensuite, le littéral positif de la clause est choisi. Nous complétons la clause en choisissant aléatoirement les littéraux négatifs.

ren$\mathcal{QHF}$ Nous commençons par choisir le nombre de variables à renommer. Ensuite, les variables à renommer sont sélectionnées avant d'engendrer les clauses par le même procédé que celui utilisé pour engendrer les $\mathcal{QHF}$s mais en renommant les littéraux sélectionnés.

Nous choisissons de n'utiliser que deux alternances de quantificateurs, $\forall X \exists Y$, tel que $|X| > \alpha$ avec $2 < \alpha < 2/3 \times \#V$ avec $\#V$ le nombre de variables de la formule.

Ces benchmarks sont triés par groupes de 1000 (resp. 100) instances de $\mathcal{QHF}$s (resp. ren$\mathcal{QHF}$s). Chaque instance a 400 variables et nous augmentons le ratio du nombre de clauses sur le nombre de variables de 3 à 6.

Une limite de temps est fixée à 60 secondes. Notons que cette limite est de 3 ordres de grandeur plus élevée que le temps de résolution de notre prouveur dédié à ce type d'instances. Des résultats similaires ont été obtenus avec une limite de temps de 300 secondes.

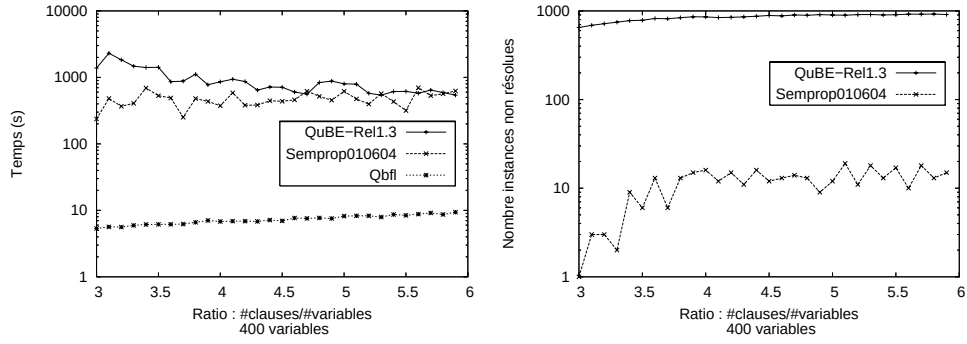


FIG. 1 – Comparaison entre Semprop, QuBE et Qbfl sur des ensembles de 1000 instances Horn aléatoires.

La figure 1 rapporte des comparaisons expérimentales sur les $\mathcal{QHF}$s. **Qbfl** étant conçu pour ce type de problèmes, il résout tous ces benchmarks très facilement. Il est plus intéressant de noter la différence de comportement des deux autres prouveurs QBF : **Semprop** peut très souvent (dans environ 99% des cas) résoudre ces problèmes rapidement (mais deux ordres de grandeur plus lentement que notre prouveur) mais échoue sur certains d'entre eux. D'un autre coté, **QuBE** ne peut résoudre la plupart de ces benchmarks. La conclusion manifeste de ces expérimentations est que la résolution

[7]Ces générateurs sont disponibles sur `http://www.cril.univ-artois.fr/~letombe/qbfg`.

d'instances de restrictions polynomiales pour $\mathcal{QBF}$ est difficile pour les prouveurs QBF actuels.

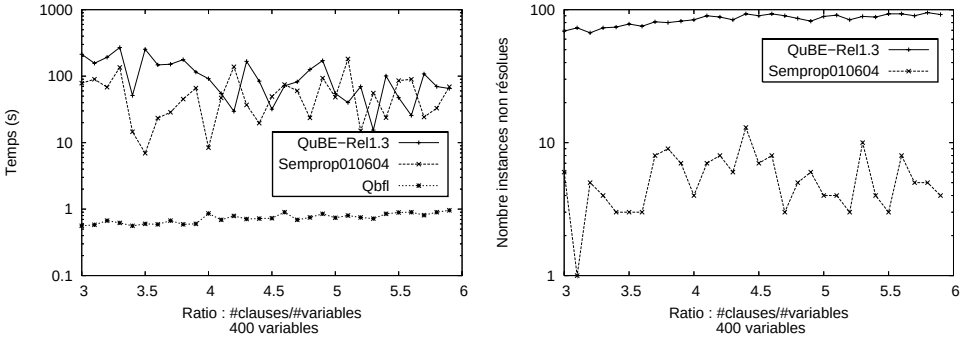


FIG. 2 – Comparaison entre Semprop, QuBE et Qbfl sur des ensembles de 100 instances Horn renommables aléatoires.

La figure 2 rapporte des comparaisons expérimentales sur les benchmarks ren$\mathcal{QHF}$s. Le comportement des prouveurs sur ce type d'instances est relativement similaire à celui sur les $\mathcal{QHF}$s. Cependant, ces benchmarks apparaissent plus difficiles pour **Semprop** qui échoue dans 10% de cas.

Notons que **Qbfl** n'apparaît pas sur le côté droit des graphes dans les figures 1 et 2 car notre prouveur a résolu tous ces benchmarks.

Observons que les temps d'exécution de **Qbfl** pour résoudre ces formules polynomiales croit lentement et régulièrement.

Ces premières expérimentations ne montrent pas que notre approche est utile : ils révèlent uniquement que notre prouveur peut détecter et résoudre des formules $\mathcal{QHF}$ et ren$\mathcal{QHF}$ efficacement et que ces formules ne sont pas triviales pour les prouveurs QBF actuels. Nous avons soumis à l'évaluation $\mathcal{QBF}$ 2005 les benchmarks pour lesquels **Semprop** a échoué.

Il est plus intéressant de voir de quelle manière le comportement de notre prouveur évolue sur l'ensemble des benchmarks de l'évaluation de l'année passée.

5.3 Benchmarks de l'évaluation $\mathcal{QBF}$ 2004

Nous nous focalisons dans un premier temps sur les formules proposées par G. Pan, traduites de la logique modale K de TANCS'98 (Balsiger *et al.*, 2000). Ces 378 instances - 9 types de 21 benchmarks valides and 9 non valides - ont été proposées pour l'évaluation de prouveurs $\mathcal{QBF}$ 2003 et la plupart d'entre eux ont été classés parmi les instances *difficiles* de l'évaluation. Les instances difficiles de cette évaluation sont celles résolues par au plus un prouveur. Depuis 2003, des prouveurs comme Quantor (Biere, 2004) se sont améliorés et ces benchmarks ne sont plus classés parmi les instances difficiles.

Les résultats obtenus par **Qbfl** sur ces instances sont présentés dans le tableau 1. Des résultats plus détaillés sont donnés en annexe. Pour chaque type d'instance, la colonne

Type d'instance	Jeroslow-Wang		Horn renommable Δ	
	%résolus	%RH	%résolus	%RH
k_branch_n	4.76	25.26	**9.52**	9.29
k_d4_n	4.76	13.25	4.76	5.63
k_dum_n	4.76	11.69	**23.80**	11.51
k_grz_n	0	-	**61.90**	11.94
k_lin_n	9.52	20.00	9.52	24.26
k_path_n	9.52	5.47	**14.28**	12.12
k_ph_n	23.80	2.66	23.80	11.73
k_poly_n	9.52	0	**14.28**	6.66
k_t4p_n	4.76	11.37	4.76	26.62
Total valides	7.93	11.21	**18.51**	13.30
k_branch_p	4.76	24.56	4.76	9.33
k_d4_p	9.52	26.33	**14.28**	25.34
k_dum_p	4.76	11.71	**14.28**	11.40
k_grz_p	0	-	0	-
k_lin_p	9.52	11.88	**19.04**	8.65
k_path_p	14.28	7.43	**19.04**	7.64
k_ph_p	19.04	10.06	19.04	6.89
k_poly_p	4.76	12.91	**9.52**	11.04
k_t4p_p	0	-	**4.76**	26.02
Total faux	7.40	14.98	**11.64**	13.28
Total	7.67	13.09	**15.07**	13.29

TAB. 1 – Pourcentages de benchmarks résolus et de ren$\mathcal{QHF}$s atteintes.

« %résolus » contient le taux d'instances résolues parmi l'ensemble des instances de ce type et la colonne « %HR » contient les taux de formules Horn renommables atteintes parmi toutes les vérifications accomplies. Ces résultats sont présentés pour **Qbfl** muni de l'heuristique de Jeroslow-Wang ainsi que l'heuristique Δ.

Les résultats complets sont présentés dans les tableaux 3 à 6. Pour chaque heuristique étudiée ici (Jeroslow-Wang et Δ), les colonnes indiquent :

Type vrai si le benchmark est valide, faux sinon,

#Branch nombre de choix de variables réalisés par l'heuristique,

#T (resp.#F) nombre de formules Horn renommables vraies (resp. fausses) atteintes durant la résolution,

Temps le temps nécessaire à **Qbfl** pour résoudre l'instance.

La principale observation est que beaucoup plus de benchmarks (environ le double) sont résolus par notre prouveur à l'aide de l'heuristique Δ par rapport à l'heuristique de Jeroslow-Wang. Le résultat est impressionnant, spécialement pour les instances *k_grz_n* : 30 benchmarks sont résolus avec l'heuristique de Jeroslow-Wang alors que 59 sont résolues avec Δ. Malheureusement, il est difficile de savoir si ces bons résultats sont dus à la détection de formules Horn renommables ou si le changement d'heuristique en est le seul responsable. En effet, nous n'avons aucun moyen de comparer le nombre de formules Horn renommables au cours de la recherche entre les deux heuristiques : réduire l'espace de recherche réduit également le nombre de formules Horn renommables détectées.

Nous avons testé Δ sur d'autres types de formules pour lesquelles le comportement de Δ n'est que peu altéré. Dans la multitude d'instances proposées par A. Ayari (72), C. Castellini (169), M. Mneimneh et K. A. Sakallah (202), J. Rintanen (67) et C. Scholl et B. Becker (64)[8], nous résolvons un total de 283 instances à l'aide des deux heuristiques (119 valides et 164 non valides). Seules 14 instances différencient les deux heuristiques : 264 avec Jeroslow-Wang et 250 à l'aide de Δ. En prenant compte des benchmarks de Pan, **Qbfl**, équipé de Δ, fonctionne mieux avec que sans. Ces résultats sont résumés dans le tableau 2.

Instance type	Total	Jeroslow-Wang			Renommable Horn Δ		
		%solved	%RH	%tps	%solved	%RH	%tps
DFlipFlopUnSat	10	100.00	0	**19.49**	100.00	0	92.08
SzymanskiPSat	12	**100.00**	9.52	0.47	75.00	10.67	0.47
ToiletASat	21	**100.00**	14.28	**0.13**	71.42	20.00	78.51
ToiletAUnSat	56	69.64	0	19.05	**85.71**	0	**10.65**
ToiletCSat	29	**100.00**	10.34	**0.00**	65.51	15.78	76.39
ToiletCUnSat	56	82.14	0	17.15	**94.64**	0	**3.09**
ToiletGSat	7	100.00	14.28	0.00	100.00	14.28	0.00
s27	4	**50.00**	94.61	**0.00**	25.00	97.30	0.73
s3271	21	100.00	0	0.24	100.00	0	0.24
ChainSat	12	66.66	99.69	80.06	**75.00**	99.72	**34.14**
ImplSat	10	**100.00**	0	**0.00**	70.00	0	12.21
LognUnSat	4	50.00	50.00	0.00	50.00	50.00	0.00
R3cnfSat	13	**100.00**	7.71	**4.18**	92.30	0.02	7.30
R3cnfUnSat	7	**100.00**	0.00	**32.59**	71.42	0.01	322.15
ToiletSat	5	**100.00**	0.00	**0.01**	40.00	0.00	26.45
ToiletUnSat	3	66.66	0	**18.70**	66.66	0	28.84
C432	8	25.00	0	**0.00**	25.00	0	0.01
C499	8	25.00	0	213.27	25.00	0	0.71
C880	8	0	-	-	**12.50**	1.51	-
comp	8	**87.50**	0.73	**3.83**	50.00	0	180.65
term1	8	50.00	0.78	**0.43**	50.00	1.87	1.24
z4ml	8	100.00	0	0.00	100.00	0	0.00
Total	334	**76.94**	8.08	**19.50**	72.74	8.73	41.71

TAB. 2 – Pourcentages de benchmarks résolus et de ren$\mathcal{QHF}$s atteintes. **%tps** est le temps moyen de résolution des instances résolues à la fois par Jeroslow-Wang et Δ.

6 Conclusion et perspectives

Notre travail porte sur l'utilisation pratique de restrictions traitables de $\mathcal{QBF}$ dans des prouveurs. Nous avons proposé une heuristique basée sur l'algorithme de détection de formules Horn renommables proposé par Hébrard. Nous avons noté qu'en pratique, les prouveurs QBF actuels ne sont pas capables de résoudre en un temps raisonnable certaines instances de $\mathcal{QHF}$ ou ren$\mathcal{QHF}$. Quantor (Biere, 2004), qui est un prouveur relativement différent des autres prouveurs QBF, n'arrive pas non plus à résoudre ces instances en temps raisonnable.

[8]Tous ces benchmarks sont disponibles sur `http://www.qbflib.org`.

En revanche, nous avons testé un prouveur de type Chaff (Moskewicz *et al.*, 2001) développé au CRIL (MiniLearning) et le prouveur Kcnfs (Dequen & Dubois, 2003) sur les mêmes benchmarks (sans le préfixe) : ils peuvent les résoudre facilement. MiniLearning peut même résoudre aisément des instances de cent mille variables.

On peut expliquer cette grande complexité du cas QBF par la contrainte imposée par l'ordre des variables dans le préfixe que la plupart des prouveur tentent de suivre en priorité. Nous nous intéressons au comportement du prouveur de Audemard et Saïs (Audemard & Saïs, 2004) sur ces benchmarks : leur prouveur ne considère pas cette contrainte. Malheureusement, les premiers résultats expérimentaux montrent que même ce prouveur ne peut résoudre facilement ces instances.

Remerciements

Merci aux relecteurs anonymes pour leurs commentaires. Ce travail a bénéficié du soutien de l'IUT de Lens et de l'Université d'Artois.

Références

ASPVALL B., PLASS M. & TARJAN R. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, **8**, 121–123. Erratum : Information Processing Letters 14(4) : 195 (1982).

AUDEMARD G. & SAÏS L. (2004). SAT based BDD solver for Quantified Boolean Formulas. In *ICTAI'04*, p. 82–89, Boca Raton (USA).

BALSIGER P., HEUERDING A. & SCHWENDIMANN S. (2000). A benchmark method for the propositional modal logics K, KT, S4. *Automated Reasoning*, **24**(3), 297–317.

BENEDETTI M. (2004). *sKizzo, A QBF decision Procedure Based On Propositional Skolemization And Symbolic Reasoning*. Rapport interne, ITC-Irst, Institute for Scientific and Technological Research, Italy.

BESNARD P., SCHAUB T., TOMPITS H. & WOLTRAN S. (2002). Paraconsistent Reasoning via Quantified Boolean Formulas, I : Axiomatising Signed Systems. In *JELIA'02*, p. 320–331, Cosenza (Italy).

BIERE A. (2004). Resolve and Expand. In *SAT'04*, p. 238–246, Vancouver (Canada).

CADOLI M., GIOVANARDI A. & SCHAERF M. (1998). An algorithm to Evaluate Quantified Boolean Formulae. In *AAAI'98*, p. 262–267, Madison (USA).

CADOLI M., GIOVANARDI A., SCHAERF M. & GIOVANARDI M. (2002). An algorithm to Evaluate Quantified Boolean Formulae and its Experimental Evaluation. *Journal of Automated Reasoning*, **28**(2), 101–142.

DAVIS M., LOGEMANN G. & LOVELAND D. (1962). A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, **5**(7), 394–397.

DAVIS M. & PUTNAM H. (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, **7**(3), 201–215.

DEQUEN G. & DUBOIS O. (2003). *kcnfs : an efficient solver for random k-SAT formulae*. Rapport interne LaRIA-2003-06, Paris VI-CNRS.

EGLY U., EITER T., TOMPITS H. & WOLTRAN S. (2000). Solving advanced reasoning tasks using quantified boolean formulas. In *AAAI'00*, p. 417–422, Austin (USA).
FARGIER H., LANG J. & MARQUIS P. (2000). Propositional Logic and One-stage Decision Making. In *KR'00*, p. 445–456, Breckenridge (CO).
FELDMANN R., MONIEN B. & SCHAMBERGER S. (2000). A distributed algorithm to evaluate quantified boolean formula. In *AAAI'00*, p. 285–290, Austin (USA).
FREEMANN J. W. (1995). *Improvement to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania.
GENT I. P. & ROWLEY A. (2002). Solving 2-CNF Quantified Boolean Formulae using Variable Assignment and Propagation. In *QBF'02*, p. 17–25, Cincinnati (USA).
GHASEMZADEH M., KLOTZ V. & MEINEL C. (2004). *ZQSAT : a QSAT Solver based on Zero-Suppressed Binary Decision Diagrams*. Rapport interne, FB-IV Informatik, University of Trier.
GIUNCHIGLIA E., NARIZZANO M. & TACCHELLA A. (2001a). Backjumping for Quantified Boolean Logic Satisfiability. In *IJCAI'01*, p. 275–281, Seattle (USA).
GIUNCHIGLIA E., NARIZZANO M. & TACCHELLA A. (2001b). QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *IJCAR'01*, p. 364–369.
HÉBRARD J. J. (1994). A Linear Algorithm for Renaming a Set of Clauses as a Horn Set. In *Theoretical Computer Science*, volume 124, p. 343–350.
JEROSLOW R. J. & WANG J. (1990). Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, **1**, 167–188.
KLEINE-BÜNING H., KARPINSKI M. & FLÖGEL A. (1995). Resolution for quantified boolean formulas. *Information and Computation*, **117**(1), 12–18.
LE BERRE D., NARRIZANO M., SIMON L. & TACHELLA A. (2005). The second QBF Solvers Comparative Evaluation. In *SAT'04*, LNCS. to appear.
LE BERRE D., SIMON L. & TACCHELLA A. (2003). Challenges in the QBF Arena : the SAT'03 evaluation of QBF solvers. In *QBF'03*, p. 468–485, S. Margherita (Italy).
LETZ R. (2002). Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *TABLEAUX'02*, Copenhagen (Denmark).
MOSKEWICZ M. W., MADIGAN C. F., ZHAO Y., ZHANG L. & MALIK S. (2001). Chaff : Engineering an Efficient SAT Solver. In *DAC'01*, p. 530–535.
PAN G., SATTLER U. & VARDI M. Y. (2002). BDD-Based Decision Procedures for K. In *CADE'02*, p. 16–30, Copenhagen (Denmark).
PAN G. & VARDI M. (2003). Optimizing a BDD-Based Modal Solver. In *CADE'03*, p. 75–89, Miami Beach (USA).
PAN G. & VARDI M. (2004). Symbolic Decision MakingProcedures for QBF. In *CP'04*, p. 453–467, Toronto (Canada).
RINTANEN J. (1999a). Constructing Conditional Plans by a Theorem-Prover. *Journal of Artificial Intelligence Research*, **10**, 323–352.
RINTANEN J. (1999b). Improvements to the Evaluation of Quantified Boolean Formulae. In *IJCAI'99*, p. 1192–1197, Stockholm (Sweden).
RINTANEN J. (2001). Partial implicit unfolding in the Davis-Putnam procedure for Quantified Boolean Formulae. In *QBF'01*, p. 84–93, Siena (Italy).
ZHANG L. & MALIK S. (2002). Towards a symetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *CP'02*, p. 200–215, Ithaca (USA).

Annexe

Instance	Jeroslow-Wang						Horn renommable Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_dum_n-1	vrai	1985397	232096	0	44.51	11.69	vrai	145638	16767	0	5.53	11.51
k_dum_n-2	-	-	-	-	-	-	vrai	327192	37689	0	11.40	11.51
k_dum_n-3	-	-	-	-	-	-	vrai	3518506	405224	0	150.34	11.51
k_dum_n-4	-	-	-	-	-	-	vrai	17139520	1973956	0	693.29	11.51
k_dum_n-5	-	-	-	-	-	-	vrai	10361259	1196794	0	517.77	11.51
k_dum_p-1	faux	1535321	179913	0	27.47	11.71	faux	92308	10599	0	2.82	11.48
k_dum_p-2	-	-	-	-	-	-	faux	1471511	167384	0	50.33	11.37
k_dum_p-3	-	-	-	-	-	-	faux	1471511	167384	0	51.29	11.37

TAB. 3 – k_dum

Instance	Jeroslow-Wang						Horn renommable Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_grz_n-1	-	-	-	-	-	-	vrai	45429	5711	0	1.78	12..57
k_grz_n-2	-	-	-	-	-	-	vrai	8454473	984272	0	392.57	11.64
k_grz_n-3	-	-	-	-	-	-	vrai	65785	8192	0	3.32	12.45
k_grz_n-4	-	-	-	-	-	-	vrai	115506	14336	0	6.83	12.41
k_grz_n-5	-	-	-	-	-	-	vrai	165943	20480	0	9.30	12.34
k_grz_n-6	-	-	-	-	-	-	vrai	308794	36862	0	21.87	11.93
k_grz_n-7	-	-	-	-	-	-	vrai	463423	55294	0	37.98	11.93
k_grz_n-8	-	-	-	-	-	-	vrai	502222	59904	0	42.58	11.92
k_grz_n-9	-	-	-	-	-	-	vrai	850324	101375	0	67.40	11.92
k_grz_n-10	-	-	-	-	-	-	vrai	1930866	222768	0	188.81	11.53
k_grz_n-11	-	-	-	-	-	-	vrai	2499281	288286	0	249.08	11.53
k_grz_n-12	-	-	-	-	-	-	vrai	1818058	209662	0	191.76	11.53
k_grz_n-13	-	-	-	-	-	-	vrai	3182241	366910	0	348.03	11.52

TAB. 4 – k_grz

Instance	Jeroslow-Wang						Horn renommable Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_lin_n-1	vrai	5	2	0	0.00	40.00	vrai	5	2	0	0.00	40.00
k_lin_n-2	vrai	2445981	0	0	102.22	0.00	vrai	259533	22162	0	18.89	8.53
k_lin_p-1	faux	3720	464	0	0.11	12.47	faux	571	113	0	0.03	19.78
k_lin_p-2	faux	3301816	372984	0	118.26	11.29	faux	147	0	0	0.01	0
k_lin_p-3	-	-	-	-	-	-	faux	1082712	160799	0	135.91	14.85
k_lin_p-5	-	-	-	-	-	-	faux	67228	0	0	11.78	0

TAB. 5 – k_lin

Instance	Jeroslow-Wang						Horn renommable Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_ph_n-1	vrai	0	1	0	0.00	100	vrai	0	1	0	0.00	100
k_ph_n-2	vrai	15	2	0	0.00	13.33	vrai	15	2	0	0.00	13.33
k_ph_n-3	vrai	181	0	0	0.00	0.00	vrai	101	17	0	0.01	16.83
k_ph_n-4	vrai	785	0	0	0.03	0.00	vrai	1647	240	0	0.11	14.57
k_ph_n-5	vrai	13119	0	0	1.12	0.00	vrai	37407	3044	0	4.22	8.13
k_ph_n-6	vrai	290163	0	0	45.15	0.00	vrai	487594	28375	0	82.33	5.81
k_ph_p-1	faux	3	1	0	0.00	33.33	faux	1	0	0	0.00	0.00
k_ph_p-2	faux	18	1	0	0.00	5.55	faux	6	0	0	0.00	0.00
k_ph_p-3	faux	535	2	0	0.01	0.37	faux	403	68	0	0.03	16.87
k_ph_p-4	faux	67297	677	0	1.00	3.56	faux	57154	6117	0	4.82	10.70

TAB. 6 – k_ph

Hybridation des méthodes de résolution pour SAT

Olivier Fourdrinoy

Centre de Recherche en Informatique de Lens, CNRS FRE 2499
Université d'Artois
Rue Jean Souvraz SP 18 F-62307 Lens Cedex
fourdrinoy@cril.univ-artois.fr

Résumé : Dans cet article nous traitons du problème SAT en général et de l'hybridation des méthodes de résolution en particulier. Déjà abordée dans différents travaux (9) cette idée consiste souvent à associer méthodes complètes et méthodes incomplètes afin d'optimiser la résolution du problème SAT en tirant partie des deux approches respectives. Nous proposons ici une nouvelle approche utilisant la recherche locale pour ordonner variables et clauses afin de guider un algorithme complet classique dans ses affectations.

Mots-clés : Logique propositionnelle, Méthode complète, Recherche Locale, Hybridation,SAT.

1 Introduction

Contrairement aux algorithmes énumératifs qui parcourent de façon systématique l'espace de recherche, les méthodes basées sur la recherche locale considèrent des « configurations », c'est-à-dire des instanciations complètes de toutes les variables de la formule. Au départ, ce sont des méthodes d'optimisation (on peut par exemple chercher à maximiser le nombre de clauses satisfaites). Sauf adaptation, ces méthodes sont incomplètes, et ne garantissent donc pas l'obtention d'une solution. En particulier, elles ne permettent pas en général de prouver qu'une instance est insatisfiable. Le principe de base des méthodes de recherche locale consiste à se déplacer judicieusement dans l'espace des configurations en améliorant la configuration courante. Ce type d'approche est généralement très efficace pour résoudre des instances satisfiables de grande taille. Nous étudions ici les possibilités de coopération entre la recherche locale et une méthode complète. Différents schémas de coopération sont observés.

Dans cet article, nous utilisons la recherche locale de deux manières. La première, assez classique puisque qu'elle est utilisée pour éventuellement trouver une solution au problème. La seconde est utilisée lorsque aucun modèle n'est trouvé pour ordonner les clauses et les variables en fonction de leurs apparitions dans les clauses falsifiées au cours de la recherche. A cette approche introduite dans (9) nous apportons des améliorations via différentes modifications notamment sur le paramétrage du nombre d'appels

à la recherche locale et sur la recherche locale elle-même.

Dans un premier temps, nous définissons quelques notions de logique propositionnelle avant de présenter brièvement les principales méthodes de résolution du problème SAT. Nous terminons cette introduction par une taxinomie d'hybridations déjà envisagées pour la résolution du problème SAT. Dans une seconde partie nous présentons notre nouvelle approche avant d'analyser quelques résultats significatifs. Enfin nous concluons en apportant les perspectives envisageables.

2 Le problème SAT

Dans cette partie, nous présentons succinctement la logique propositionnelle puis le problème SAT en développant les deux principales méthodes de résolution.

2.1 Définitions préliminaires

Soit $\mathcal{B}$ un langage booléen (i.e. propositionnel) de formules, utilisant les connecteurs usuels($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) et un ensemble de variables propositionnelles.

Une *formule CNF* Σ est un ensemble (interprété comme une conjonction) de *clauses*, où une clause est un ensemble (interprété comme une disjonction) de *littéraux*. Un littéral est une variable propositionnelle positive ou négative. On note $\mathcal{V}(\Sigma)$ (resp. $\mathcal{L}(\Sigma)$) l'ensemble des variables (resp. littéraux) apparaissant dans Σ. Une *clause unitaire* est une clause formée d'un unique littéral. Un *littéral unitaire* est l'unique littéral d'une clause unitaire.

Une *interprétation* d'une formule booléenne est une affectation des valeurs $\{vrai, faux\}$ de ces variables. Un *modèle* d'une formule CNF est une interprétation qui satisfait la formule. Le problème SAT revient à décider si une formule CNF admet un modèle ou à prouver qu'elle n'en admet pas.

Rappelons que toute formule propositionnelle admet une forme CNF qui lui est équivalente.

La plupart des démonstrateurs automatiques exploitent le théorème de déduction. Ils utilisent au moins un test de satisfaisabilité pour montrer qu'une information peut être déduite à partir d'une base de connaissances. Le problème SAT s'est révélé important au moins à ce titre. Par ailleurs, il occupe également une place particulière et centrale parmi les problèmes NP-complets. Toutes ces raisons ont fait que, au cours de ces 50 dernières années, les recherches portant sur le problème SAT ont été nombreuses et variées. De nombreux algorithmes ont notamment été proposés pour le résoudre. Ces algorithmes se décomposent principalement en deux groupes : les algorithmes complets et les algorithmes incomplets.

2.2 Les méthodes complètes

Les algorithmes complets répondent à la fois à la satisfiabilité et à l'insatisfaisabilité. On les dit « complets » car si on leur laisse un temps et un espace illimité, ils parcourent *complètement* l'espace de recherche. Il existe deux principaux types d'algorithmes complets. Les algorithmes syntaxiques et les algorithmes sémantiques. Les

premiers s'intéressent à la structure du problème en ne tenant pas compte des valeurs des variables, on prendra ici pour exemple le principe de résolution. Les seconds s'attachent au sens des variables pour tenter de trouver une solution, ils ont pour principal représentant la méthode de (4).

2.2.1 Le principe de résolution

Le principe de résolution est donc un « algorithme syntaxique ». C'est un principe élémentaire mais fondamental en logique. On applique successivement les trois règles suivantes jusqu'à l'arrêt de la procédure :

- règle de résolution
- règle de fusion
- règle de sous-sommation

Si on arrive sur la clause vide, c'est à dire que la clause générée ne contient plus de littéraux on peut conclure qu'il n'y a pas de solution.

Par contre si l'on ne parvient plus à générer de nouvelles clauses et que l'on n'a pas généré de clause vide, c'est qu'il existe au moins une solution.

Le principe de résolution n'est pas l'algorithme complet le plus utilisé du fait que le nombre de clauses à ajouter dans le pire des cas est exponentiel par rapport à la taille initiale du problème.

La méthode présentée dans (5) propose de choisir une à une les variables et à chaque fois de générer toutes les résolvantes liées à cette variable. Ensuite on peut supprimer toutes les clauses contenant cette variable. Au final, soit on trouve la clause vide et on conclut à l'insatisfaisabilité, soit on ne parvient plus à générer de résolvantes et on conclut à la satisfaisabilité de l'instance. Cependant, la génération d'un modèle est quelque peu fastidieuse et cet algorithme a tendance à saturer la mémoire. En effet, en cours d'exécution, le nombre de clauses générées peut vite s'avérer très important.

2.2.2 La procédure de Davis Logemann et Loveland

La méthode développée par Davis, Logemann, et Loveland appelée parfois DPLL ou encore plus abusivement DP (Putnam apparaissant pour des raisons historiques) paraît beaucoup plus naturelle puisqu'il s'agit à priori de parcourir l'ensemble des solutions de manière systématique. Le *Backtrack* et la *propagation unitaire* ainsi que le traitement des littéraux purs [1] sont les composantes essentielles des procédures de type Davis et Putnam.

L'algorithme de Davis et Putnam fonctionne ainsi. Il nettoie à chaque étape l'ensemble de clauses. C'est-à-dire qu'il élimine systématiquement, les clauses unitaires et les clauses où apparaissent un littéraux purs et supprime les occurences de tous les littéraux opposés aux littéraux unitaires traités. Une fois ce nettoyage effectué, et s'il n'a pas donné lieu à la preuve de la consistance ou de l'inconsistance de la formule, l'algorithme poursuit sa recherche dans l'arbre en affectant la variable de son choix. Ce choix est l'un des points importants de l'algorithme de Davis et Putnam. C'est d'ailleurs une

[1] Un littéral est dit pur dans une CNF si son opposé n'apparaît dans aucune clause de la CNF.

Algorithm 1: Algorithme de Davis et Putnam

fonction *DPLL* **return** *boolean*
Data: un ensemble Σ de clauses
Result: true si Σ est consistant, false sinon
begin
  Σ^* =Propagation_Unitaire(Σ);
  if Σ^* *contient la clause vide* **then** return false;
  else
    Σ^+ =Simplification_Littéraux_Purs(Σ^*);
    if $\Sigma^+ == \emptyset$ **then** return true;
    else
      l=Heuristique_de_branchement(Σ^+);
      return $(DPLL(\Sigma^+ \cup \{(l)\}))\|(DPLL(\Sigma^+ \cup \{(\neg l)\}))$;
end

des principales variations entre les différents algorithmes qui s'en sont inspirés. Une fois le choix fait, DP est appelé récursivement avec les deux sous-formules générées.

De nombreuses améliorations de cet algorithme ont été proposées ces dernières années. Parmi elles on notera les *Watched litterals* dans (11) qui offrent une nouvelle structure de données plus adaptée à la propagation unitaire ainsi que les travaux sur les *dépendances fonctionnelles* dans (12) qui utilisent un pré-traitement pour analyser la structure des problèmes posés.

2.3 Les méthodes incomplètes

Les algorithmes incomplets, par définition, ne parcourent pas *complètement* l'arbre de recherche. En général, ils ne répondent pas à la question de l'insatisfaisabilité.

2.3.1 La recherche locale

Une méthode de recherche locale n'effectue pas un parcours systématique de l'espace de recherche. Le plus souvent elle choisit une affectation des variables du problème au hasard et ensuite, si la solution n'est pas trouvée, on flippe (c'est à dire qu'on inverse sa valeur) les variables une à une en suivant une stratégie de réparation. L'algorithme s'arrête lorsqu'une solution est trouvée ou lorsque le temps ou l'espace dédié à l'algorithme est dépassé.

La méthode de choix de la variable à *flipper* caractérise la méthode de recherche locale. L'objectif de ces méthodes est de s'extraire des *minima locaux*. Il s'agit de situations où quelque soit la variable que l'on flippe, le nombre de clauses insatisfaites ne diminue pas. Il est nécessaire de trouver des heuristiques (dites d'échappement) qui permettent de sortir de ce genre de piège.

Pour cet algorithme, il faut dégager différents points stratégiques : tout d'abord dans les entrées, le nombre de réparations maximal n'est pas anodin. Il est parfois préfé-

Algorithm 2: Algorithme de recherche locale

```
fonction RL return boolean
Data: un ensemble Σ de clauses et max_reparations le nombre maximum de réparations autorisées
Result: true si Σ admet un modèle, false si on ne peut conclure
begin
    Générer une configuration initiale I;
    nb_reparations = 0;
    while (nb_reparations < max_reparations) and (I n'est pas un modèle) do
        if une descente est possible then
            Remplacer I par une interprétation voisine permettant la descente;
        else
            Remplacer I par une interprétation voisine selon le critère d'échappement ;
        nb_reparations++;
    return (I est modèle) ;
    %% retourner la valeur du test : "I est un modèle ?"
end
```

rable de recommencer plusieurs fois l'algorithme à zéro plutôt que de s'enliser dans un nombre trop grand de réparations.

Ensuite la génération de la configuration initiale peut se faire de différentes manières. Il est par exemple possible de choisir une répartition 50/50 des littéraux vrais et des littéraux faux, ou encore mettre tout à vrai ou tout à faux. Ces choix peuvent s'avérer important selon la nature du problème.

Enfin, vient le choix de la variable à « flipper » qui s'effectue dans la boucle *while*. Certains algorithmes de recherche locale prônent une analyse totale du problème pour choisir celle qui aura le meilleur rendement. D'autres, pour gagner du temps, préfèrent prendre une variable au hasard. Les meilleures solutions s'appuient sur un compromis entre ces deux extrêmes.(Selman *et al.*; 8)

2.4 Les méthodes hybrides

La notion d'hybridation est ici un abus de langage. Les algorithmes de résolution pour SAT sont complets ou incomplets. La notion d'hybridation est dérivée du fait que l'on greffe généralement entre eux des algorithmes complets et incomplets. Le résultat est un algorithme complet ou incomplet selon la nature de la greffe. Mais en aucun cas il ne s'agit d'un troisième statut.

Dans le cas de SAT, l'hybridation revient d'un coté à utiliser des heuristiques inspirées de DPLL pour la recherche locale et réciproquement des heuristiques inspirées de la recherche locale pour DPLL. Les résultats donnent donc des méthodes complètes ou incomplètes.

La collaboration entre les méthodes systématiques et les méthodes non systématiques constitue l'un des challenges proposés par Selman(14). L'idée de ce challenge est de montrer que la coopération des deux méthodes est plus efficace que chacune d'elles prises séparément.

Pour présenter ces méthodes dites *hybrides*, nous les classons par souche. Par exemple, si on utilise une recherche locale pour trouver la variable de branchement d'une approche systématique, on considère que la souche est de type complète (ou systématique).

2.4.1 Schémas de coopération basés sur une souche incomplète

Nous trouvons ici les méthodes de résolution qui se basent sur une méthode incomplète. Notons que cette approche a surtout été explorée dans le domaine des CSP. Comme il est écrit dans un paragraphe précédent, on peut caractériser une recherche locale par l'ensemble des opérations suivantes :

- le choix d'une assignation initiale des variables ;
- le choix de la prochaine assignation ;
- un critère d'arrêt.

Pour chacune de ces opérations, il est possible d'utiliser des informations issues des méthodes complètes. Nous présentons une liste non-exhaustive de méthodes hybrides basées sur des méthodes incomplètes.

– **résolution alternée :** Zhang & Zhang (16) effectuent une assignation initiale partielle des variables. La recherche continue par un algorithme de *backtrack* (type DPLL) qui tente de résoudre le sous problème induit. Le processus est ensuite réitéré. On parle ici de *résolution alternée* car les méthodes travaillent chacune leur tour.

– **résolution de sous problème par DP :** Lorsque l'on se trouve confronté à un noyau de clauses difficilement satisfiables, on peut tenter de résoudre le sous-problème de manière systématique. Cette approche peut permettre par ailleurs de prouver l'inconsistance ce qui n'est pas le cas d'une méthode incomplète classique. En réalité cette méthode est plutôt exploitée dans l'autre sens, c'est à dire que l'on utilise la recherche locale pour détecter des noyaux inconsistants (2) et y choisir les variables de branchement pour une recherche systématique.

– **recherche locale complète :** Plus récemment(Fang & Ruml), H.Fang et W.Ruml ont proposé un nouvel algorithme du nom de *cls* (pour *Complete Local Search*) qui tout en se basant sur la recherche locale offre la particularité d'être complet. L'algorithme *cls* effectue une recherche locale avec pour heuristique d'échappement l'ajout de clauses en utilisant le principe de résolution. Ainsi, soit la recherche locale donne un modèle, soit on ne peut plus générer de clause et dans ce cas il y a un modèle, soit la clause vide est générée et le problème est déclaré insatisfiable. Typiquement, cette coopération rend la recherche locale complète.

2.4.2 Schémas de coopération basés sur une souche complète

Beaucoup plus prolixe que la précédente, cette coopération cherche à exploiter les nombreuses informations fournies par la recherche locale afin de guider l'algorithme dans le parcours de l'arbre des solutions.

– **Complétude partielle :** méthode duale à la résolution alternée, la *complétude partielle* limite DP à une profondeur variable au delà de laquelle une recherche locale

est lancée. La profondeur dépend d'un ensemble de paramètres tels que le temps disponibles ou l'estimation de la meilleure solution espérée.

– **pré-traiter pour ordonner :** la recherche locale est utilisée pour un pré-traitement du problème. Elle peut fournir éventuellement une solution pour des instances SAT mais l'objectif est avant tout d'identifier les éléments les plus problématiques de l'instance.
  Il existe plusieurs alternatives suivant que l'on s'intéresse aux clauses, aux variables ou encore aux littéraux. Ensuite, la recherche systématique est guidée par les informations collectées durant la recherche locale. Concrètement on effectue un ordonnancement des variables ou des littéraux en fonction de leurs apparitions ou non dans les clauses régulièrement insatisfaites par la recherche locale. De nombreuses approches sont possibles selon que l'on utilise tel ou tel algorithme de recherche locale et suivant l'heuristique de pondération choisie.
  J.Crawford(3) expérimenta cette méthode sur les instances 3SAT aléatoires au seuil sans succès alors que B.Mazure(7) a obtenu des résultats plus probant sur un plus large panel d'instances. Une approche similaire en se basant sur les littéraux permet d'affiner l'ordonnancement des variables.
  Il est encore possible de pondérer les clauses et d'ordonnancer celles-ci. Ensuite DP prendra comme variable de choix les variables de la clause la plus falsifiée et ainsi de suite.
– **Heuristique de branchement :** L'utilisation de la recherche locale comme heuristique de branchement (9) prend en compte le fait que l'ordonnancement peut évoluer au fur et à mesure de l'exploration de l'arbre de recherche. Ainsi, au lieu de se baser sur un ordonnancement statique des variables (des clauses ou des littéraux), on recalcule régulièrement celui-ci. DPRL appelle ainsi à chaque point de choix une heuristique de type recherche locale qui si elle ne trouve pas de solution, identifie le littéral apparaissant le plus souvent dans les clauses falsifiées.
  La fréquence des appels à la recherche locale est bien entendue paramétrable et c'est d'ailleurs l'inconvénient majeur de la méthode. Une recherche locale utilise des ressources de temps non négligeable et un recours systématique à celle-ci peut s'avérer parfois coûteux.

Notre approche s'inspire par ailleurs de cette dernière idée.

3 Description de notre approche

Nous avons travaillé sur une coopération entre la recherche locale et une méthode de type DPLL. Comme nous l'avons vu précédemment, la recherche locale est généralement incapable de prouver l'inconsistance. Elle peut cependant fournir un ensemble d'informations pouvant aider à détecter cette inconsistance. Une grande partie des instances inconsistantes se caractérisent par la présence d'un ou plusieurs noyaux inconsistants. On entend par là que seul un sous-ensemble de variables (et/ou de clauses)intervient dans l'insatisfaisabilité de l'instance. Notre but est de circonscrire cet ensemble par la recherche locale pour ensuite piocher les variables de décision de la méthode systématique dans ce sous-ensemble.

3.1 Détection de noyaux inconsistants

3.1.1 Les noyaux inconsistants

Définition 1
Une CNF Σ *est* **globalement inconsistante** *si et seulement si* Σ *est inconsistante et* $\forall$ $\Pi \subset \Sigma$, Π *est consistante.*

Lorsqu'une CNF inconsistante n'est pas globalement inconsistante, elle est dite localement inconsistante.

Cependant il est possible de définir plusieurs degrés de localité. En effet cette localité peut être caractérisée par le rapport entre la taille de la plus petite sous-CNF inconsistance et la taille de la CNF initiale. Les sous-CNF inconsistantes de la base sont appelées *noyaux inconsistants*.

Définition 2
Soit Σ *une CNF inconsistance.* Π *est appelé* **noyau inconsistant** *de* Σ *si et seulement si* $\Pi \subset \Sigma$ *et* Π *est inconsistant. On dit que* Π *est* **noyau globalement inconsistant** *si et seulement si* Π *est globalement inconsistant. Un noyau inconsistant* Π *de* Σ *est dit* **minimalement inconsistant** *si et seulement si* Π *est globalement inconsistant et pour tout noyau inconsistant* Ω *de* Σ, $|\Pi| \leq |\Omega|$.

Ces différentes formes d'inconsistance présentent un intérêt majeur notamment dans les applications dérivant de SAT. L'inconsistance d'une base de connaissances est souvent due à la présence accidentelle d'une information contradictoire. La détection de ces données contradictoires est utile dans de nombreuses applications, comme par exemple la détection de pannes.

3.2 Recherche locale et détection de noyaux inconsistants

A chaque fois qu'une recherche locale propose une interprétation falsifiée, on sait qu'au moins une clause de chaque noyau inconsistant est présente dans les clauses falsifiées par l'interprétation courante. Cette propriété fournit un ensemble d'informations pouvant amener à détecter des noyaux inconsistants plus rapidement dans une méthode systématique. L'idée majeure est donc d'exploiter au « maximum » les informations fournies par la recherche locale dans le but de circonscrire un noyau inconsistant. En effet, un seul noyau suffit pour montrer l'inconsistance d'une CNF.

Un noyau inconsistant se compose de clauses qui elles-mêmes se composent de littéraux. On peut donc identifier un noyau inconsistant de deux manières :

– en identifiant les clauses.
– en identifiant les littéraux.

Pour ces derniers, l'inconsistance vient bien évidemment de ce que les littéraux sont présents à la fois positivement et négativement. Donc on cherche plutôt à identifier les variables du noyau.

Nous avons donc développé la méthode suivante. Nous avons marqué les clauses les plus falsifiées au cours de la recherche locale. Ainsi, à la fin de celle-ci, si aucun modèle

n'a été détecté, nous avons un ordre sur les clauses qui permet de guider la méthode systématique dans ses choix de variables de décision.

Cependant, la recherche locale se caractérise par une phase de « descente ». En effet, après avoir affecté aléatoirement les variables, on cherche à diminuer le nombre de clauses falsifiées. Ce nombre de clauses fausses initiales peut s'avérer important sans que cela soit représentatif du problème. En général, ce nombre décroît rapidement au cours des premiers *flips* avant de se stabiliser. Les informations relatives à cette descente ne sont pas à priori significatives. On pourrait aisément les assimiler à un phénomène de « bruit » dans le domaine des statistiques. Afin de ne pas perdre de temps sur ces dernières, nous avons développé deux stratégies : une basée sur un seuil l'autre sur les minima locaux.

3.2.1 Heuristique de pondération avec seuil

Définition 3
Le **seuil de pondération** *est une constante entière qui indique le nombre maximum de clauses falsifiées en dessous duquel on marque celles-ci.*

La notion de seuil se base sur le fait que la recherche locale bute en général sur un très petit nombre de clauses. Ainsi, on ne marquera que les clauses (ou les littéraux) présentes dans le cas de figure suivant : le nombre de clauses falsifiées est inférieur au seuil k fixé par l'utilisateur. Toute la difficulté réside ici dans le réglage du seuil. La valeur optimale du seuil est fortement liée à la nature de l'instance testée. Nous avons obtenu des résultats intéressants avec cette stratégie mais un problème très simple s'est posé. Si n noyaux inconsistants indépendants (qui ne possèdent pas de variables en commun) constituent le problème, il faudra que le seuil soit au moins égal à n. Or nous ne sommes pas capables de savoir à l'avance combien de noyaux constituent l'instance. Aussi avons nous développé l'heuristique suivante.

3.2.2 Heuristique de pondération basée sur les minima

En marquant les clauses présentes dans les minima locaux[2], l'algorithme évite de perdre du temps dans les « descentes » et il n'est pas tributaire du nombre de noyaux inconsistants. Cette stratégie semble plus pertinente car elle ne nécessite pas de régler le seuil. Celui-ci fluctue dynamiquement au cours de la recherche locale. Cependant, si l'heuristique d'échappement ne permet pas de se sortir d'un creuset d'insatisfaction, il peut être difficile de cerner le noyau inconsistant.

3.3 Une heuristique de branchement basée sur la recherche locale

L'idée de cette contribution est de proposer une nouvelle combinaison qui utilise la recherche locale comme heuristique de branchement pour un algorithme de type DPLL. Nous utilisons donc une version basique de l'algorithme DPLL et un algorithme de recherche locale WSAT ou « WalkSat » (13).

[2]Un minimum local est une interprétation telle que quelle que soit la variable flippée, il est impossible d'améliorer le nombre de clauses satisfaites.

Il existe encore de nombreuses options possibles quant au paramétrage de cette collaboration :

- utiliser WSAT systématiquement ;
- utiliser WSAT jusqu'à une profondeur prédéfinie puis une heuristique classique ;
- utiliser WSAT pour une profondeur proportionnelle à la taille de l'instance traitée puis une heuristique classique ;

Nous comparons ces trois approches dans la partie 4.

L'algorithme (WSAT) se singularise par sa méthode de choix de la variable à flipper. En effet, au lieu d'utiliser une heuristique qui va parcourir toutes les clauses et donc toutes les variables afin de choisir celle dont le « flip » paraît le plus utile, WSAT prend une clause falsifiée au hasard, et se concentre sur les variables de cette clause. Cette méthode surprenante présente un gain de temps non négligeable. Par exemple, sur une instance 3-SAT, on ne regardera que 3 variables au maximum même si l'instance compte des milliers de variables.

Pour chacune des variables contenues dans la clause falsifiée, on va attribuer un « score » et la variable de plus grand score sera choisie pour être inversée.

La fonction Score fonctionne très simplement en calculant le nombre de clauses qui seront falsifiées en cas de flip de la variable, puis le nombre de clauses qui seront satisfaites par ce même flip et en renvoyant la différence entre ces deux valeurs. On préférera bien évidemment les variables favorisant l'augmentation du nombre totale de clauses satisfaites.

3.4 Intégration de l'heuristique dans DPLL

Après la recherche locale, nous obtenons un tableau contenant les scores des clauses et nous pouvons donc établir un classement des clauses les plus falsifiées. Nous pouvons également établir un classement des variables les plus souvent présentes dans les clauses falsifiées. En utilisant ce second classement, il est possible d'affecter en priorité les variables posant le plus de problème. Cependant, dès qu'une affectation est faite, les clauses les plus difficilement satisfiables sont validées et il peut être intéressant de relancer la recherche locale afin de voire quelles sont dès lors les nouvelles clauses « difficiles ».

3.4.1 Fréquence de l'appel à la recherche locale

Nous avons suite à différentes expérimentations sur les instances aléatoires fixé le paramétrage suivant. La recherche locale est appelée tant que la profondeur de l'arbre d'exploration des solutions est inférieure à 5. Ensuite, on se base sur le dernier classement des variables obtenus pour choisir les variables d'affectations.

3.4.2 Diversification de la recherche locale

Lors de nos premiers tests, nous choisissions l'interprétation initiale au hasard (en prenant soin de ne pas toucher aux variables déjà affectées dans DPLL). Une amélioration consiste à construire la nouvelle interprétation initiale à partir des dernières affectations connues des variables à affecter. En effet, ces affectations résultant soit d'un

backtrack dans DPLL, soit de la dernière recherche locale, cela permet de se concentrer sur des affectations apparemment défectueuses. Puis afin de diversifier les terrains d'explorations de la recherche locale, nous effectuons un flip général sur toutes les variables (concernées par le recherche locale). En somme, la recherche locale ressasse les derniers problèmes rencontrés puis envisage les configurations opposées.

L'idée est que nous n'utilisons pas la recherche locale dans son but premier (trouver une solution). Aussi nous souhaitons explorer le maximum d'interprétations différentes et cette méthode prétend diversifier les recherches.

3.4.3 Synthèse

- tant que la profondeur de l'arbre est inférieure à 5 :
  - WSAT + heuristique de pondération basée sur les minima locaux ;
  - affectation et propagation de la variable de meilleur score ;
- affectation et propagation des variables en suivant le dernier classement fournit par WSAT.

4 Résultat expérimentaux et analyses

Nos expérimentations ont eu pour but de valider expérimentalement notre approche en la comparant à des approches plus classiques.

Pour ces expérimentations, nous avons utilisé les 4 prouveurs suivants :

- un DPLL avec 1 appel à la recherche locale appelé DP_1_RL (9) ;
- une recherche locale RL : WSAT ;
- un DPLL avec accès systématique à la recherche locale DP_ALL (9) ;
- notre prouveur décrit précédemment.

Nous avons comparé les approches sur deux types d'instances : les instances aléatoires issues du modèle classique de génération et les instances AIM issues de DIMACS(1)

Les expérimentations ont été réalisées sur des P3 fonctionnant sous linux Fedora core 2.

Les résultats détaillés en annexe correspondent au temps moyen en seconde mis pour résoudre les instances. Un TIME OUT de 1000 secondes a cependant été fixé.

Les résultats expérimentaux montrent que les utilisations unique, ou au contraire systématique de la recherche locale comme heuristique de branchement ont de moins bon résultats que l'utilisation partielle de celle-ci. Il est évident que la recherche locale utilisée systématiquement coûte très cher, surtout sur les instances insatisfiables. De plus l'utilisation unique (au lancement) de la recherche locale patit d'un défaut majeur de la recherche locale. En effet, celle-ci a tendance à explorer des parties relativement petites de l'espace de recherche. On risque donc de baser DPLL sur un cas particulier de l'espace de recherche, d'autant que pour simplifier notre paramétrage, nous avons fixé le nombre de flips.

Les résultats obtenus sont conformes à nos attentes. Les différentes approches présentent des résultats cohérents. En effet la recherche locale est plus efficace que DP sur les instances SAT.

instances	DP_1_RL	RL	DP_ALL	notre prouveur
SAT	145.13	111.37	153.31	85.27
UNSAT	306.48	#	325.7	136.41

TAB. 1 – tableau de synthèse

DP_1 et DP_ALL montrent certaines limites. On peut supposer que DP_1 paye le manque de réactivité de son heuristique. En effet une fois la recherche locale terminée, il utilise constamment le classement fourni et ne met jamais à jour les données qu'il contient. Par contre DP_ALL semble pâtir du trop grand nombre d'appels à la recherche locale.

Un appel mesuré à la recherche locale semble porter ses fruits sur les instances UNSAT, ce que nous espérions mais également sur les instances SAT où notre approche surpasse la recherche locale. Nous pensons que ces résultats sont dus au fait que notre heuristique permet alors de cerner non pas les noyaux inconsistants, mais ce qu'on pourrait appeler les noyaux de difficulté. Des sous-ensembles de clauses présentant un petit nombre d'interprétations possibles.

5 Conclusion et perspectives

Dans ce papier nous apportons une nouvelle approche de coopération entre la recherche locale et les méthodes systématiques de résolution pour SAT. Les résultats présentés montrent un comportement intéressant même s'ils se limitent à des algorithmes très basiques. La lourdeur du paramétrage de telles collaborations nous à amener à faire des choix arbitraires et à utiliser en premier lieu de tels algorithmes. La pertinence de ces résultats nous encourage à porter ce travail vers des solveurs plus aboutis.

Il pourrait également être intéressant d'explorer les notions de « heavy-tail » (6), de « backbone » (10) ou encore de « backdoor » (15) qui constituent à l'instar des noyaux inconsistants d'autres façons de caractériser le "noyau dur" d'un problème.

Références

(1993). : Center for Discrete Mathematics and Computer Science of Rutgers University.

CHVTAL V. & SZEMEREDI E. (1988). Many hard examples for resolution. *Journal of the ACM*.

CRAWFORD & AUTON (1993). Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93)*, p. 21–27.

DAVIS, LOGEMANN & LOVELAND (1962). A machine program for theorem-proving. *Journal of the Association for Computing Machinery*, **5**, 394–397.

DAVIS & PUTNAM (1960). A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, (7), 201–215.

FANG & RUML. Complete local search for propositional satisfiability.
GOMES C. P., SELMAN B., CRATO N. & KAUTZ H. (2000). Heavy-tail phenomena in satisfiability and constraint satisfaction. *Journal of Automated Reasoning*.
MAZURE (1999). *De la satisfaisabilité à la compilation de bases de connaissances propositionnelles*. PhD thesis, université d'artois.
MAZURE, SAÏS & GRÉGOIRE (1997). Tabu search for sat. In *Fourteenth National Conferece on Artificial Intelligence*, p. 281–285, Providence(Rhode Island, USA).
MAZURE, SAÏS & GRÉGOIRE (1998). Boosting complete techniques thanks to local search. *Annals of Mathematics and Artificial Intelligence*, **22**, 309–322.
MONASSON R., ZECCHINA R., KIRKPATRICK S., SELMAN B. & TROYANSKY L. (1999). Determining computational complexity from characteristic 'phase transitions'. *Nature*.
MOSKEWICZ, CONOR, MADIGAN, ZHAO, ZHANG & MALIK (2001). Chaff : Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*.
OSTROWSKI, GRÉGOIRE, MAZURE & SAIS (2002). Recovering and exploiting structural knowledge from cnf formulas. In *Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002)*, p. 185–199, Ithaca (N.Y.).
SELMAN, KAUTZ & COHEN (1994). Noise strategies for improving local search. In *Twelfth National Conference on Artificial Intelligence(AAAI'94)*, p. 337–343.
SELMAN, KAUTZ & MCALLSTER (1997). Computational challenges in propositional reasoning and search. In *Fifteenth International Joint Conference on Artificial Intelligence(IJCAI'97)*, p. 50–54, Nagoya(Japan).
SELMAN, LEVESQUE & MITCHELL. Gsat : A new method for solving hard satisfiability problems. In *the tenth National Conference on Artificial Intelligence*, p. 440–446 : AAAI'92.
WILLIAMS R., GOMES C. P. & SELMAN B. (2003). Backdoors to typical case complexity. In *Proc. of IJCAI'03*.
ZHANG & ZHANG (1995). Sem : a system for enumerating models. p. 298–303.

Annexe

instances	DP_1_RL	RL	DP_ALL	notre prouveur
aim_50_yes	0	0.39	0.05	0.01
aim_50_no	0.02	#	0.11	0.06
aim_100_yes	126.51	13.28	140.89	0.11
aim_100_no	529.28	#	750.1	0.24
aim_200_yes	485.04	516.51	569.13	502.17
aim_200_no	1000	#	1000	557.39

TAB. 2 – résultats comparatifs sur les AIM

instances	nbVar	nbCla	DP_1_RL	RL	DP_ALL	notre prouveur
al325	50	163	0.1	0.14	0.1	0.12
al325	100	325	0.1	0.13	0.1	0.14
al325	150	488	0.1	0.15	0.1	0.14
al325	200	650	0.09	0.14	0.1	0.12
al325	250	813	0.25	0.14	0.13	0.15
al325	300	975	0.11	0.17	0.4	0.15
al425s	50	213	0.01	0.07	0.01	0.01
al425s	100	425	0.03	4.81	0.04	0.55
al425s	150	638	1.25	82.18	1.22	2.19
al425s	200	850	41.48	243.21	100.86	6.33
al425s	250	1063	656.38	270.36	644.33	79.5
al425s	300	1275	865.45	538.94	887.23	687.31
al425u	50	213	0.01	#	0.01	0.07
al425u	100	425	0.08	#	0.13	1.29
al425u	150	638	1.71	#	4.77	5.63
al425u	200	850	102.75	#	276.16	42.86
al425u	250	1063	937.22	#	1000	239.95
al425u	300	1275	1000	#	1000	949.38
al525	50	263	0.01	#	0	0.03
al525	100	525	0.05	#	0.03	0.44
al525	150	788	0.31	#	0.42	3.28
al525	200	1050	5.55	#	10.43	17.46
al525	250	1313	92.56	#	204.41	61.97
al525	300	1575	867.64	#	965.84	166.1

TAB. 3 – résultats comparatifs sur les aléatoires

Fusion locale de contraintes temporelles avec priorité

Mahat Khelfallah, Belaïd Benhamou

LSIS - UMR CNRS 6168, CMI Université de Provence
39, rue Joliot-Curie, 13453 Marseille Cedex 13
{mahat,benhamou}@cmi.univ-mrs.fr

Résumé : L'information provient souvent de plusieurs sources et la fusion de ces dernières provoque habituellement l'apparition d'incohérences. La fusion est l'opération qui consiste à restaurer la cohérence des informations fusionnées en gardant le maximum d'informations initiales inchangées. Dans ce papier, nous nous intéressons à la fusion de contraintes linéaires avec priorité, dans le cadre des problèmes temporels simples (STPs). La priorité exprime une relation de préférence définie sur les contraintes linéaires, et peut représenter soit le degré de confiance, le niveau de qualité des contraintes, ou la fiabilité de leurs sources. Nous étendons le formalisme de STP avec des priorités et nous proposons une méthode de fusion locale, que nous expérimentons sur des instances de STP avec priorité générées aléatoirement.

Mots-clés : Fusion, Contraintes temporelles, Problèmes temporels simples STPs, Priorités

1 Introduction

L'information provient habituellement de plusieurs sources et la fusion de ces dernières provoque souvent l'apparition d'incohérences. La fusion consiste à restaurer la cohérence des informations fusionnées en gardant le maximum d'informations initiales inchangées.

La fusion d'informations est une branche très importante de l'intelligence artificielle, et plusieurs travaux de recherche ont été menés dans ce domaine. Mais la majorité de ces travaux ont été effectués dans le cadre de la logique (propositionnelle, possibiliste, etc.) (Lin & Mendelzon, 1999; Benferhat *et al.*, 1997; Konieczny *et al.*, 2002). La fusion d'informations munies de priorités a également suscité l'intérêt des chercheurs (Benferhat *et al.*, 2002; Meyer *et al.*, 2001).

Par ailleurs, il existe plusieurs domaines d'applications où les informations peuvent être représentées par des contraintes linéaires. Par exemple, les problèmes d'ordonnancement (Kolisch & Padman, 2001), certaines informations géographiques peuvent être également exprimées par des contraintes linéaires (Kuper *et al.*, 2000; Rigaux *et al.*, 2002; Khelfallah & Benhamou, 2004a; Khelfallah & Benhamou, 2004b).

Dans ce papier, nous nous intéressons à la fusion des contraintes linéaires munies de priorités dans le cadre des problèmes temporels simples (STPs). Nous supposons

que chaque STP provient d'une source d'information. Nous étendons le formalisme de STP pour prendre en compte les priorités qui expriment une relation de préférence définie sur les contraintes. Cette préférence peut provenir de deux situations : *(i)* Toutes les sources d'information ont la même fiabilité, par conséquent, tous les STPs ont la même priorité. Mais les contraintes de chaque STP sont ordonnées suivant leur qualité ou leur degré d'importance. *(ii)* Les sources d'information sont ordonnées suivant leur fiabilité ou leur degré de qualité. Ceci définit un pré-ordre total sur les STPs. Cependant, les contraintes de chaque STP ont la même priorité. Dans les deux situations, l'union de tous les STPs génèrera un *STP avec priorité* qui est probablement incohérent. La restauration de sa cohérence consiste en deux étapes principales : la détection de ses conflits et leur élimination.

Dans un précédent travail (Khelfallah & Benhamou, 2004a; Khelfallah & Benhamou, 2004b), des méthodes de révision de contraintes linéaires ont été proposées dans le cadre d'une application géographique réelle (application de l'inondation). Ce papier en constitue une généralisation dans la mesure où : (1) des contraintes linéaires plus générales ont été considérées. En effet, le problème d'inondation a été représenté par un STP particulier où seules les contraintes impliquant la variable origine étaient corrigeables, et (2) des priorités définies sur les contraintes ont été prises en compte, ce qui constitue la principale contribution de ce papier.

Dans (Benferhat & Garcia, 2002), différentes stratégies de résolution locales de conflits dans des bases de croyances incohérentes ont été proposées. Le principe de ces méthodes consiste à identifier tous les conflits (ensembles minimaux incohérents de formules) de la base de croyances, puis à établir une relation de précédence sur les conflits, en vue de définir des priorités sur eux. Cette relation de précédence est basée sur des relations d'influence définies sur les conflits et traduisant l'interdépendance pouvant exister entre les conflits. Les stratégies de résolution proposées par la suite corrigent ces conflits. Comme nous le verrons dans ce papier, la méthode que nous proposons est différente de celles proposées dans (Benferhat & Garcia, 2002), dans la mesure où d'abord nous ne déterminons pas l'ensemble de tous les conflits, opération qui peut s'avérer impossible vu la grande complexité de cette opération. Puis, aucun ordre n'est établi entre les conflits. Et contrairement à la stratification des formules ou sources dans (Benferhat & Garcia, 2002), la stratification que nous considérons dans ce papier est établie au début et ne dépend pas du contexte. Une telle démarche est certes restrictive, mais elle a l'avantage d'être plus intuitive et plus facile à mettre en oeuvre.

Le reste du papier est organisé comme suit. Dans la section 2, nous rappelons quelques préliminaires sur les problèmes temporels simples STPs. Nous étendons dans la section 3, le formalisme de STP avec des priorités et nous présentons le principe de la fusion. Nous proposons dans la section suivante une méthode de fusion. Cette méthode est expérimentée sur des instances de STPs avec priorité, et les résultats obtenus sont donnés dans la section 5, avant de conclure dans la section 6.

2 Préliminaires

Un problème temporel simple (en anglais, *Simple Temporal Problem* STP) S est défini par S=$(\mathcal{X}, C)$ où $\mathcal{X}$ est un ensemble fini de variables X_0, ..., X_n, définies sur des

domaines continus. Ces variables représentent des événements temporels (des points de temps) et X_0 représente l'origine du temps. C est l'ensemble des contraintes de la forme $X_j - X_i \leq a_{ij}$ définies sur ces variables, où a_{ij} est un scalaire. Chaque contrainte exprime une distance entre deux événements temporels. Les contraintes de la forme $X_j - X_i \geq a_{ij}$ peuvent être également représentées puisque $X_j - X_i \geq a_{ij}$ est équivalente à $X_i - X_j \leq -a_{ij}$.

Un tuple $x = (x_1, ..., x_n)$ de valeurs réelles est une *solution* du STP S si l'instantiation $\{X_1{=}x_1, ..., X_n{=}x_n\}$ satisfait toutes ses contraintes. Le STP S est *cohérent* si et seulement si il possède une solution.

Nous associons au STP $S{=}(\mathcal{X}, C)$ un graphe orienté et pondéré $G_d = (\mathcal{X}, E_d)$, appelé le *graphe de distances* où $\mathcal{X}$, l'ensemble des sommets, est l'ensemble des variables du STP S, et E_d est l'ensemble des arcs pondérés représentant les contraintes de C. Chaque contrainte $X_j - X_i \leq a_{ij}$ de C est représentée par l'arc $i \rightarrow j$[1], pondéré par a_{ij}. Pour plus de détails, voir (Dechter *et al.*, 1991).

3 Fusion de contraintes avec priorité

Avant de présenter la fusion de contraintes avec priorité, nous étendons le formalisme de STP en considérant des priorités. Nous définissons d'abord la notion d'un ensemble de contraintes avec priorité.

Définition 1
Un ensemble de contraintes avec priorité est un ensemble de contraintes $\mathcal{C}$ stratifié en r strates $\mathcal{C}^1, \ldots, \mathcal{C}^r$ (i.e., $\mathcal{C} = \mathcal{C}^1 \cup \ldots \cup \mathcal{C}^r$, et $\forall i, j,\ i \neq j : \mathcal{C}^i \cap \mathcal{C}^j = \emptyset$), où toutes les contraintes de chaque strate $\mathcal{C}^i$ ont la même priorité et ont une plus haute priorité que les contraintes de $\mathcal{C}^j$ pour tout j tel que $i < j \leq r$.
Un STP avec priorité S est une paire $S = (\mathcal{X}, \mathcal{C})$ où $\mathcal{X}$ est un ensemble de variables et $\mathcal{C}$ est un ensemble de contraintes avec priorité.

La fusion de plusieurs STPs avec priorité peut provoquer l'apparition de conflits même si chacun des STPs avec priorité considérés est initialement cohérent. Soient $S_1{=}(\mathcal{X}_1, \mathcal{C}_1)$,..., $S_p{=}(\mathcal{X}_p, \mathcal{C}_p)$ p STPs avec priorité issus de p différentes sources d'information. Et soit $S = (\mathcal{X}, \mathcal{C})$ le STP avec priorité obtenu à partir des STP avec priorité S_1,..., S_p, où $\mathcal{X} = \bigcup_{1 \leq i \leq p} \mathcal{X}_i$ et $\mathcal{C} = \bigcup_{1 \leq i \leq p} \mathcal{C}_i$. Nous pouvons distinguer les situations suivantes :

- Les p STPs avec priorité ont le même degré de fiabilité. Nous supposons que les priorités sur les contraintes dans les STPs avec priorité fusionnés sont commensurables et que pour chaque STP avec priorité $S_i = (\mathcal{X}_i, \mathcal{C}_i)$, l'ensemble des contraintes $\mathcal{C}_i$ est stratifié en r strates $\mathcal{C}_i^1, \ldots, \mathcal{C}_i^r$. L'ensemble de contraintes $\mathcal{C}$ est stratifié en r strates $\mathcal{C}^1, \ldots, \mathcal{C}^r$ où chaque ensemble de contraintes $\mathcal{C}^k$ (ayant la priorité k) est l'union des ensembles de contraintes $\mathcal{C}_i^k$ ayant la priorité k pour chaque STP avec priorité S_i, i.e., $\mathcal{C}^k = \bigcup_{1 \leq i \leq p} \mathcal{C}_i^k$.

[1]Pour simplifier, un sommet X_i du graphe de distances G_d est dénoté par son indice i.

– Les p STPs sont ordonnés selon leur fiabilité mais les contraintes de chaque STP S_i sont de priorités égales. $\mathcal{C}$ est stratifié en r ($r \leq p$) strates $\mathcal{C}^1, \ldots, \mathcal{C}^r$ où chaque $\mathcal{C}^k$ est l'union des ensembles de contraintes des STPs ayant la priorité k,
$\mathcal{C}^k = \bigcup_{S_i=(\mathcal{X}_i,\mathcal{C}_i),\ S_i \text{ a la priorité } k} \mathcal{C}_i$.
– Les p STPs avec priorité sont ordonnés selon leur fiabilité et pour chaque STP avec priorité $S_i = (\mathcal{X}_i, \mathcal{C}_i)$, l'ensemble de contraintes $\mathcal{C}_i$ est stratifié en r_i strates. Dans ce cas, il faut combiner les deux priorités : la priorité sur les STPs, et la priorité sur les contraintes d'un même STP. Pour cela, plusieurs solutions existent, par exemple, favoriser la priorité sur les STPs, ou au contraire, favorise la priorité sur les contraintes. La solution que nous choisissons dans ce travail tient compte des deux priorités, et est définie comme suit : l'ensemble des contraintes $\mathcal{C}$ du STP avec priorité S est stratifié en r strates $\mathcal{C}^1, \ldots, \mathcal{C}^r$ telles que pour toute contrainte c de $\mathcal{C}$, si $c \in \mathcal{C}_i^j$, si c est une contrainte de priorité j, appartenant au STP avec priorité i qui a une priorité k, alors c aura la priorité $j \times k$ dans le STP avec priorité S, c'est à dire, $c \in \mathcal{C}^{j \times k}$.

Remarquez que le dernier cas est une généralisation des deux premiers. Lorsque les STPs ont le même degré de fiabilité, nous retrouvons la premième situation, et lorsque les contraintes dans chaque STP ont la même priorité, nous retrouvons la deuxième situation.

Sans perte de généralité, nous supposons dans ce qui suit que S contient au plus une contrainte pour chaque paire ordonnée de variables[2]. Si le STP avec priorité S est cohérent, la fusion est faite. Dans le cas contraire, la cohérence de S doit être restaurée.

Exemple 1
Soient $S_1 = (\mathcal{X}_1, \mathcal{C}_1)$ et $S_2 = (\mathcal{X}_2, \mathcal{C}_2)$ deux STPs où $\mathcal{X}_1 = \{X_0, X_1, X_2, X_3\}$, $\mathcal{C}_1 = \{X_0 - X_1 \leq -40,\ X_3 - X_1 \leq 10\ , X_2 - X_0 \leq -30\ , X_1 - X_2 \leq 10\}$, $\mathcal{X}_2 = \{X_0, X_2, X_3, X_4\}$ et $\mathcal{C}_2 = \{X_0 - X_2 \leq 20\ , X_2 - X_4 \leq -45,\ X_4 - X_3 \leq 20\ , X_2 - X_3 \leq -25\}$. Nous supposons que la source du STP S_1 est plus fiable que celle du STP S_2, ce qui entraîne la primauté des contraintes de S_1 sur celles de S_2. L'union des STPs S_1 et S_2 donne le STP avec priorité $S = (\mathcal{X}, \mathcal{C})$ où $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 = \{X_0, X_1, X_2, X_3, X_4\}$ et $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. Les contraintes de $\mathcal{C}_1$ ont la plus haute priorité qui est égale à 1 et les contraintes de $\mathcal{C}_2$ ont la priorité 2.

La notion de priorité n'est pas utilisée dans la définition du graphe de distances d'un STP avec priorité, qui reste la même que celle du graphe de distances d'un STP classique.

Nous manipulons, tout au long de ce papier, le STP avec priorité S au lieu des STPs avec priorité S_1, ..., S_p. Soient n , m et r respectivement les nombres de variables, de contraintes et de priorités dans le STP avec priorité S. Le STP avec priorité S est défini par $S = (\mathcal{X}, \mathcal{C})$ tel que $\mathcal{C} = \mathcal{C}^1 \cup \ldots \cup \mathcal{C}^r$, où $\mathcal{C}^i$, $1 \leq i \leq r$, est l'ensemble des contraintes de priorité i tel que 1 est la plus haute priorité, et r la plus basse. Soit G_d le graphe de distances associé au STP avec priorité S. Donc, n et m sont respectivement le

[2] S'il existe deux contraintes $X_i - X_j \leq a_{ij}$ et $X_i - X_j \leq b_{ij}$ dans le STP avec priorité S telles que $a_{ij} < b_{ij}$, alors seule la contrainte $X_i - X_j \leq a_{ij}$ est considérée dans S.

nombre de sommets et le nombre d'arcs dans le graphe de distances G_d. Faute d'espace, les preuves des théorèmes et propositions donnés dans ce papier sont omises.

3.1 Détection des conflits

La première étape de l'opération de restauration de la cohérence du STP avec priorité S est la détection de ses conflits. La méthode que nous proposons est basée sur une variante du résultat suivant.

Théorème 1
(Shostak, 1981; Lia & Wong, 1983; Leiserson & Saxe, 1983) Un STP est cohérent si et seulement si son graphe de distances ne contient pas de circuits négatifs[3].

Nous pouvons déduire du théorème 1 que pour restaurer la cohérence d'un STP, il faut supprimer tous les circuits négatifs de son graphe de distances. En réalité, il suffit de supprimer tous les circuits élémentaires[4] négatifs du graphe de distances. Ceci affaiblit la condition du théorème 1. Nous étendons ce résultat aux STPs avec priorité et nous obtenons le théorème suivant sur lequel est basé notre méthode de fusion.

Théorème 2
Un STP avec priorité est cohérent si et seulement si son graphe de distances ne contient pas de circuits élémentaires négatifs.

La présence de circuits élémentaires négatifs dans G_d est synonyme de présence de conflits dans le STP avec priorité S. Un conflit est défini comme suit.

Définition 2
Soit S un STP avec priorité et G_d son graphe de distances. Un conflit de S est un couple (σ, w) où σ est un circuit élémentaire négatif du graphe de distances G_d et w est le poids[5] *du circuit σ.*

Pour détecter les conflits du graphe des distances, nous définissons une procédure, appelée *Détection-Conflits*, qui est une extension de l'algorithme de *Bellman-Ford* pour la recherche des plus courts (par rapport au poids) chemins dans un graphe (Cormen *et al.*, 1990). L'idée principale est de calculer pour chaque paire de sommets (i, j), du graphe de distances G_d, le plus court chemin allant de i vers j. En particulier, lorsque $i = j$, la procédure calcule le plus court circuit passant par le sommet i.

La procédure *Détection-Conflits* est donnée par l'algorithme 1. Elle consiste en deux étapes. Dans la première, elle construit une matrice $mat^{(0)}$ dont les éléments sont les paires définies par : $mat^{(0)}_{ij} = (p_{ij}, w_{ij})$ où p_{ij} représente un chemin de longueur 1 (arc) allant de i vers j dans le graphe G_d, et w_{ij} est le poids du chemin p_{ij}. La phase d'initialisation est terminée en copiant la matrice $mat^{(0)}$ dans la matrice mat. La seconde étape est une boucle qui calcule les plus courts chemins entre chaque paire de

[3] Un circuit négatif est un circuit dont la somme des poids de ses arcs est négative.

[4] Un circuit élémentaire est un circuit qui ne contient aucun autre circuit.

[5] Le poids d'un chemin est la somme des poids de ses arcs.

sommets (i, j). À chaque itération l de la boucle *tant que*, la fonction *Extension-Plus-Courts-Chemins*, donnée dans l'algorithme 2, est appelée afin de calculer les plus courts chemins de longueur l entre chaque paire de sommets. La boucle s'arrête soit lorsqu'un conflit est détecté, ou bien lorsque la longueur des chemins calculés atteint n.

Algorithme 1
Procédure *Détection-Conflits(* G_d,$Conf$*)*
Var $mat^{(0)}$*,* mat *: matrices de paires (chemin,poids)*
Début
{ Initialisation }
$Conf := \emptyset$
$l := 2$
pour $i, j := 1$ *à* n **faire**
s'*il existe un arc* $i \rightarrow j$ *dans* G_d *pondéré par* a_{ij} **alors** $mat_{ij}^{(0)} := ((i, j), a_{ij})$
sinon si $i \neq j$ **alors** $mat_{ij}^{(0)} := (\emptyset, \infty)$ **sinon** $mat_{ij}^{(0)} := (\emptyset, 0)$
$mat := mat^{(0)}$
{ Extension des chemins }
tant que *(* $l \leq n$ **et** $Conf = \emptyset$*)* **faire**
début
$mat :=$ Extension-Plus-Courts-Chemins*(*$mat^{(0)}, mat, Conf$*)*
$l := l + 1$
fin
Fin

La fonction *Extension-Plus-Courts-Chemins* est basée sur le principe suivant : le plus court chemin p_{ij}, de longueur l, allant de i vers j est composé d'un plus court chemin p_{ik} de longueur $l - 1$ allant de i vers k et d'un arc allant de k vers j.

Algorithme 2
Fonction Extension-Plus-Courts-Chemins*(*$mat^{(0)}$*,* mat*,* **Var** $Conf$*) : la matrice de (chemin,poids) étendue*
Var mat' *la matrice de (chemin,poids) étendue*
Début
{ Initialisation }
pour $i, j := 1$ *à* $n, i \neq j$ **faire** $mat'_{ij} := (\emptyset, \infty)$
pour $i := 1$ *à* n **faire** $mat'_{ii} := (\emptyset, 0)$
{ Extension des chemins de mat }
pour $i, j := 1$ *à* n **faire**
pour $k := 1$ *à* n **faire**
si *(*$mat_{ik}.chemin \neq \emptyset$ **et** $mat_{kj}^{(0)}.chemin \neq \emptyset$ **et**
$mat_{ik}.poids + mat_{kj}^{(0)}.poids < mat'_{ij}.poids$*)* **alors**
début
$mat'_{ij} := (mat_{ik}.chemin \bullet mat_{kj}^{(0)}.chemin, mat_{ik}.poids + mat_{kj}^{(0)}.poids)$
si *(*$i = j$*)* **alors** $Conf := Conf \cup mat'_{ii}$
fin
Extension-Plus-Courts-Chemins *:=* mat'
Fin

Lorsque la fonction *Extension-Plus-Courts-Chemins* est appelée à l'itération l de la boucle de la procédure *Détection-Conflits*, elle prend comme arguments : $mat^{(0)}$ la matrice initiale des paires (chemin,poids) correspondant au graphe de distances G_d, mat la matrice des paires (chemin,poids) représentant les plus courts chemins, de longueur $l-1$, dans G_d. Elle retourne la matrice mat' des plus courts chemins de longueur l, dans G_d. En particulier, mat'_{ii} contiendra un plus court circuit négatif de longueur l (s'il existe) passant par le sommet i. De plus, tous les circuits négatifs détectés sont élémentaires, et sont rajoutés dans $Conf$. La fonction *Extension-Plus-Courts-Chemins* retourne dans $Conf$ l'ensemble des conflits dont les circuits négatifs sont de longueur l. Si $Conf = \emptyset$, alors il n'existe pas de circuits négatifs de longueur l dans G_d.

Nous évaluons maintenant la complexité de la fonction *Extension-Plus-Courts-Chemins*. La phase d'initialisation est effectuée en $O(n^2)$. La seconde phase consiste en trois boucles. Chaque itération de la boucle la plus interne peut être exécutée en $O(n)$ puisque les tests sur les chemins et les poids sont effectués en un temps constant et la concaténation des chemins est faite en $O(n)$. Donc, la seconde phase peut être effectuée en $O(n^4)$. Par conséquent, la complexité de la fonction *Extension-Plus-Courts-Chemins* est $O(n^4)$ dans le pire des cas.

Nous évaluons à présent la complexité de la procédure *Détection-Conflits*. La phase d'initialisation peut être effectuée en $O(n^2)$. Dans la seconde phase, la procédure effectue au plus $n-1$ itérations, et la complexité de chacune d'entre elles est identique à la complexité de la procédure *Extension-Plus-Courts-Chemins* qui est $O(n^4)$. Par conséquent, la complexité de la procédure *Détection-Conflits* est $O(n^5)$ dans le pire des cas.

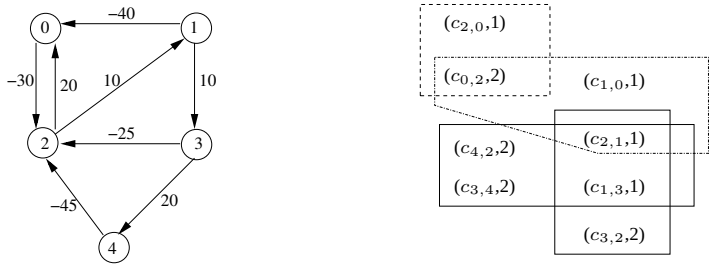
3.2 Représentation des conflits

Chaque conflit du STP avec priorité S est identifié par une paire (σ, w) où σ est un circuit élémentaire négatif du graphe de distances G_d et w est le poids de σ. Nous rappelons que chaque arc $i \rightarrow j$ de G_d, pondéré par a_{ij}, représente la contrainte $c_{ij} : X_j - X_i \leq a_{ij}$ du STP avec priorité S. Nous définissons la notion de contrainte conflictuelle.

Définition 3
Soient $S = (\mathcal{X}, C)$ un STP avec priorité et G_d son graphe de distances. Une contrainte $c_{ij} \in C$ est une contrainte conflictuelle s'il existe un conflit $c = (\sigma, w)$ de S tel que l'arc $i \rightarrow j$ appartient au circuit élémentaire négatif σ de G_d.

Soit $ConfConst$ la fonction qui associe à chaque conflit $c = (\sigma, w)$ l'ensemble de ses contraintes conflictuelles. Formellement, $ConfConst(c) = \{c_{ij} \in C : i \rightarrow j$ est un arc de $\sigma\}$.

L'ensemble des conflits détectés $Conf$ est représenté par un hypergraphe $H_c = (V, E_c)$ à sommets pondérés où V est l'ensemble des sommets pondérés correspondant à l'ensemble de toutes les contraintes conflictuelles défini par : $V = \bigcup_{c \in Conf} ConfConst(c)$, et E_c est l'ensemble des hyper-arêtes défini comme suit : chaque hyper-arête e représente un conflit c de $Conf$ qui lui-même est représenté par ses contraintes conflictuelles, i.e.,



a. Le graphe de distances b. L'hypergraphe de conflits

FIG. 1 – Le graphe de distances et l'hypergraphe des conflits du STP avec priorité S défini dans l'exemple 1

$e = ConfConst(c)$[6]. Donc, $E_c = \{ConfConst(c) : c \in Conf\}$. Chaque sommet c_{ij} de V est pondéré par k, où k est la priorité de la contrainte c_{ij} (i.e., $c_{ij} \in \mathcal{C}^k$). H_c est appelé l'*hypergraphe des conflits* du STP avec priorité S. L'ensemble des sommets V est stratifié en r strates $V_1, \ldots, V_r$ telles que : $v \in V_i$ si le poids du sommet v dans l'hypergraphe H_c est i.

Exemple 2
Le graphe de distances du STP avec priorité S défini dans l'exemple 1 est représenté dans la figure 1.a. Le circuit élémentaire négatif {(0,2),(2,0)} défini un conflit entre les contraintes $c_{0,2}$ et $c_{2,0}$. Ceci ajoute l'hyper-arête $\{(c_{0,2}, 2), (c_{2,0}, 1)\}$ dans l'hypergraphe des conflits, où chaque sommet représente une contrainte pondérée par sa priorité, par exemple, la contrainte $c_{0,2}$ pondérée par 2. En considérant tous les circuits élémentaires négatifs du graphe de distances de la figure 1.a, on obtient l'hypergraphe des conflits de la figure 1.b.

3.3 Identification d'un sous-ensemble de contraintes à corriger

La suppression de tous les conflits détectés nécessite la correction de certaines contraintes impliquées dans ces conflits. Plus précisément, au moins une contrainte conflictuelle de chaque conflit détecté doit être corrigée. Autrement dit, l'intersection de l'ensemble des contraintes corrigées et l'ensemble des contraintes conflictuelles de chaque conflit est non vide. Par conséquent, l'ensemble des contraintes corrigées définit un *transversal* de l'hypergraphe des conflits H_c du STP avec priorité S. Minimiser le changement lors de la restauration de la cohérence d'un STP avec priorité S revient à minimiser le nombre de contraintes corrigées de $\mathcal{C}^1$ (ayant la plus haute priorité), puis le nombre de contraintes corrigées de $\mathcal{C}^2$, etc. Nous rappelons la définition d'un transversal avant de définir la notion de *transversal prioritaire*.

Définition 4
Soit $H = (V, E)$ un hypergraphe. T est un transversal *de l'hypergraphe H si $T \subseteq V$ et*

[6]Nous rappelons qu'une hyper-arête est définie comme un sous-ensemble de sommets de l'hypergraphe.

pour toute hyper-arête e de E, $T \cap e \neq \emptyset$.

Nous définissons une relation de préférence sur les transversals d'un hypergraphe à sommets pondérés.

Définition 5
Soit $H = (V, E)$ un hypergraphe à sommets pondérés où V est stratifié en r strates $V_1, \ldots, V_r$ telles que le sommet $v \in V_i$ si v est pondéré par i dans H. Soient T_1 et T_2 deux transversals de l'hypergraphe H. T_1 est préféré à T_2 si (1) $\exists i$, $1 \leq i \leq r$ tel que $|T_1 \cap V_i| < |T_2 \cap V_i|$; et (2) $\forall j$, $1 \leq j < i$, $|T_1 \cap V_j| = |T_2 \cap V_j|$.

Nous pouvons définir à présent ce qu'est un transversal prioritaire.

Définition 6
Soit $H = (V, E)$ un hypergraphe à sommets pondérés. T_p est un transversal prioritaire de l'hypergraphe H si (1) T_p est un transversal de H ; et (2) pour tout transversal T de H, T n'est pas préféré à T_p.

Exemple 3
L'hypergraphe de conflits représenté dans la figure 1.b a plusieurs transversals. Par exemple, $T = \{(c_{0,2}, 2), (c_{2,1}, 1)\}$. Il a également deux transversals prioritaires $T_{p_1} = \{(c_{0,2}, 2), (c_{3,2}, 2), (c_{3,4}, 2)\}$ et $T_{p_2} = \{(c_{0,2}, 2), (c_{3,2}, 2), (c_{4,2}, 2)\}$.

La recherche d'une couverture de sommets dans un graphe est un cas particulier de la recherche d'un transversal dans un hypergraphe. Sachant que la recherche d'une couverture de sommets de taille fixe dans un graphe est un problème NP-Complet (Garey & Johnson, 1979), la recherche d'un transversal de taille fixe dans un hypergraphe est un problème au moins NP-Complet. La recherche d'un transversal prioritaire est un problème NP-Difficile puisqu'un critère d'optimisation est rajouté. On peut réduire substantiellement cette complexité par considérant un "bon" transversal prioritaire de l'hypergraphe des conflits au lieu d'un transversal prioritaire. Nous définissons à cette fin, la fonction *Bon-Transversal-Prioritaire* (Algorithme 3) basée sur une heuristique qui considère d'abord les sommets ayant le plus grand poids (correspondant à la plus basse priorité) dans l'hypergraphe des conflits.

Soient n_c et m_c respectivement le nombre de sommets et le nombre d'hyper-arêtes de l'hypergraphe des conflits H_c. La fonction *Bon-Transversal-Prioritaire* calcule d'abord le degré de chaque sommet de l'hypergraphe des conflits H_c. La complexité de cette opération est en $O(m_c n_c)$. La fonction *Bon-Transversal-Prioritaire* effectue au plus n_c itérations puisque, dans le pire des cas, tous les sommets de l'hypergraphe H_c sont pris. À chaque itération, V_i l'ensemble de sommets ayant le plus grand poids dans l'hypergraphe est sélectionné. Ceci peut être accompli en $O(r)$ où r est le nombre de priorités considérées dans S. La sélection du sommet ayant le plus grand degré dans V_i est faite en $O(n_c)$. La suppression de toutes les hyper-arêtes incidentes au sommet v est effectuée en $O(n_c)$ et la mise à jour des degrés des sommets impliqués dans les hyper-arêtes supprimées est faite en $O(n_c m_c)$. Donc, la complexité d'une itération est $O(n_c m_c)$. Par conséquent, la complexité de la fonction *Bon-Transversal-Prioritaire* est $O(m_c n_c^2)$ dans le pire des cas.

Algorithme 3
Fonction *Bon-Transversal-Prioritaire*($H_c = (V, E_c)$) *: bon transversal de* H_c
Var T *: un ensemble de sommets de* H_c
Début
  $T := \emptyset$
  pour chaque *sommet* v *de* V **faire** *calculer* $deg(v)$ *le degré de* v
  répéter
    sélectionner la plus basse priorité i *telle qu'il existe* $V_i = \{v \in V : v$ *est pondéré par* $i\} \neq \emptyset$
    sélectionner v *le sommet ayant le plus grand degré dans* V_i
    $T := T \cup \{v\}$
    pour *chaque hyper-arête* e *de* E_c *telle que* $v \in e$ **faire**
    début
      supprimer e *de* E_c
      pour *chaque sommet* $v' \in e$ **faire** $deg(v') := deg(v') - 1$
    fin
  jusqu'à $E_c = \emptyset$
  Bon-Transversal-Prioritaire *:=* T
Fin

Exemple 4
La fonction Bon-Transversal-Prioritaire *appliquée à l'hypergraphe de conflits de la figure 1.b retourne le transversal* $T = \{(c_{0,2}, 2), (c_{3,2}, 2), (c_{4,2}, 2)\}$ *dont les trois éléments sont de priorité basse égale à 2.*

3.4 Correction des contraintes conflictuelles

Après avoir identifié les contraintes à corriger pour éliminer les conflits détectés, nous allons voir comment effectuer de telles corrections. Soit $c = (\sigma, w)$ un conflit du STP avec priorité S. L'élimination du conflit c nécessite la suppression de son circuit élémentaire négatif σ. Ceci implique la correction d'au moins une des contraintes impliquées dans le circuit σ, i.e., au moins une des contraintes de $ConfConst(c)$ doit être corrigée. La proposition suivante montre comment les contraintes sont corrigées.

Proposition 1
Soient S *un STP avec priorité et* $c = (\sigma, w)$ *un conflit de* S*. Soit* $c_{ij} : X_j - X_i \leq a_{ij}$ *une contrainte conflictuelle de* c *(*$c_{ij} \in ConfConst(c)$*). Remplacer la contrainte* $c_{ij} : X_j - X_i \leq a_{ij}$ *par la contrainte* $X_j - X_i \leq a_{ij} - w$ *suffit pour éliminer le conflit* c*.*

Exemple 5
Dans la figure 1.a, le circuit élémentaire négatif {(0,2),(2,1),(1,0)} de poids -60 identifie le conflit $c = (\{(0,2),(2,1),(1,0)\}, -60)$ *entre les contraintes* $X_2 - X_0 \leq -30$*,* $X_1 - X_2 \leq 10$ *et* $X_0 - X_1 \leq -40$*. Le remplacement de la contrainte* $X_2 - X_0 \leq -30$ *par la contrainte* $X_2 - X_0 \leq 30$ *supprime le conflit* c*.*

La correction des contraintes ne peut générer de nouveaux conflits et le théorème suivant stipule que la correction des contraintes correspondant à un transversal de l'hypergraphe des conflits représentant les conflits détectés suffit pour éliminer ces conflits.

Théorème 3
Soient S un STP avec priorité et $Conf$ l'ensemble des conflits détectés de S. Soit H_c l'hypergraphe des conflits représentant l'ensemble $Conf$. Les conflits de $Conf$ sont éliminés du STP avec priorité S si et seulement si les contraintes correspondant à un transversal de l'hypergraphe des conflits H_c sont corrigées.

Après avoir présenté les différentes étapes de la fusion, nous proposons un algorithme de fusion.

4 Algorithme de fusion locale avec priorité

Puisque le nombre de circuits élémentaires négatifs dans le graphe de distances d'un STP avec priorité est potentiellement très grand, la détection exhaustive des conflits peut s'avérer impossible. Un traitement local semble une bonne alternative à ce problème. En d'autres termes, si les conflits sont détectés et corrigés par paquets, la complexité de l'opération de fusion va considérablement baisser. Par ailleurs, si un conflit détecté c, d'un STP avec priorité S, implique par exemple une contrainte c_{ij}, et si cette contrainte participe dans un autre conflit c' qui n'a pas encore été détecté, alors la correction de la contrainte c_{ij} pourrait éliminer le conflit c'.

Nous présentons un algorithme de fusion locale (algorithme 4), appelé algorithme de *Fusion-Locale-Prior*, qui consiste à détecter un ensemble de conflits, puis à les éliminer en corrigeant les contraintes conflictuelles correspondant à un "bon" transversal prioritaire de l'hypergraphe des conflits. Il répète ces opérations jusqu'à la restauration de la cohérence.

Algorithme 4
Fonction Fusion-Locale-Prior*(S : STP avec priorité) : STP avec priorité cohérent*
Début
  construire G_d le graphe de distance associé à S
  Détection-Conflits$(G_d, Conf)$
  répéter
    construire H_c l'hypergraphe des conflits correspondant à $Conf$
    T := Bon-Transversal-Prioritaire (H_c)
    corriger les contraintes correspondant à T
    Détection-Conflits$(G_d, Conf)$
  jusqu'à $Conf = \emptyset$
  Fusion-Locale-Prior := S
Fin

Théorème 4
L'algorithme de fusion locale avec priorité, appliqué au STP avec priorité S, termine et restaure la cohérence de S.

Pour évaluer la complexité de l'algorithme *Fusion-Locale-Prior*, nous procédons étape par étape. Soit m_c le nombre de conflits du STP avec priorité S. Ce nombre est borné par le nombre de circuits élémentaires possibles dans le graphe de distances

G_d, qui est lui-même borné par $\sum_{k=2}^{n} A_n^k$ où $A_n^k = \frac{n!}{(n-k)!}$. À chaque itération, n conflits peuvent être détectés et par la suite corrigés. Ainsi le nombre d'itérations effectuées est borné par $\frac{m_c}{n}$. En pratique, le nombre d'itérations n'atteint jamais le pire des cas $\frac{m_c}{n}$, car la correction d'un conflit pourrait éliminer d'autres conflits non encore détectés.

Nous évaluons à présent la complexité de chaque itération. La complexité de la procédure *Détection-Conflits* est $O(n^5)$. Étant donné qu'au plus n conflits peuvent être détectés à chaque itération, et que chaque conflit peut impliquer au plus n contraintes conflictuelles, la construction de l'hypergraphe des conflits H_c correspondant à $Conf$ est en $O(n^2)$. Un "bon" transversal prioritaire T de l'hypergraphe de conflits H_c est calculé en $O(m_c' n_c^{'2})$ dans le pire des cas, où m_c' est le nombre de conflits traités et n_c' est le nombre de contraintes conflictuelles. Le nombre m_c' est borné par n car au plus n conflits sont considérés à chaque itération et n_c' est borné par n^2. Donc, la recherche d'un bon transversal prioritaire est accomplie en $O(n^5)$ dans le pire des cas. La correction des contraintes de T est faite en $O(n)$ car au plus n contraintes peuvent être corrigées. Donc, chaque itération est achevée en $O(n^5)$ dans le pire des cas. L'algorithme *Fusion-Locale-Prior* effectue au plus $\frac{m_c}{n}$ itérations. Par conséquent, sa complexité est $O(m_c n^4)$ dans le pire des cas.

5 Résultats expérimentaux

L'algorithme de fusion locale avec priorité présenté dans ce papier a été implanté en C et testé sur des instances aléatoires de STPs avec priorité. Le programme a été exécuté sur un PC muni d'un processeur P4 à 2.2 MHz et d'une mémoire vive de 512 Mo.

La génération aléatoire de STPs avec priorité est basée sur trois paramètres : n le nombre de variables, r le nombre de priorités (strates), et d la densité des contraintes qui est définie par le rapport du nombre de contraintes par le nombre de contraintes possibles, i.e., $d = \frac{nombre\ de\ contraintes}{n(n-1)}$. La dureté des contraintes est représentée par l'intervalle [-50,50] où les poids des contraintes sont générés. Un échantillon de 50 problèmes est généré pour tout tuple (n, r, d) et les moyennes des mesures ont été prises. Les résultats obtenus par l'application de l'algorithme *Fusion-Locale-Prior* sur des instances aléatoires de STPs avec priorité sont présentés dans le tableau 1.

Nous pouvons remarquer, dans le tableau 1, que l'augmentation de la densité entraîne l'augmentation du nombre de conflits, ce qui a pour effet, l'augmentation des nombres de contraintes corrigées (aussi bien le nombre total, que les nombre des contraintes les plus et les moins prioritaires), et le nombre d'itérations ainsi que le temps d'exécution. Lorsque le nombre de priorités considérées augmente, les nombres des conflits détectés, de contraintes corrigées, d'itérations et le temps d'exécution augmentent aussi. Comme prévu, le nombre de contraintes corrigées est minimal lorsque seule une priorité est considérée (autrement dit, lorsque toutes les contraintes ont la même priorité). Ceci est dû au principe de changement minimal. Nous pouvons remarquer que l'algorithme réussit à fusionner, en un temps raisonnable, des problèmes à 200 variables.

		$\mathbf{n} = 50$			$\mathbf{n} = 100$			$\mathbf{n} = 200$		
		Densité			**Densité**			**Densité**		
	r	**0.2**	**0.5**	**0.8**	**0.2**	**0.5**	**0.8**	**0.2**	**0.5**	**0.8**
nb conf.	**1**	491	1058	1467	2026	4067	5744	7651	16086	23998
	5	700	1506	2045	2977	5614	7668	10772	21236	29847
	10	714	1848	2516	3752	7172	9330	12882	24077	34933
nb cont.*	**1**	281	707	1032	1296	2984	4393	5483	12643	19024
	5	612	1379	1888	2710	5195	7173	9976	20027	28403
	10	643	1735	2391	3503	6875	8970	11321	23223	33959
nb cpp*	**1**	281	707	1032	1296	2984	4393	5483	12643	19024
	5	10	100	187	154	576	924	914	2735	4251
	10	0	21	63	24	225	411	312	1212	1974
nb cmp*	**1**	281	707	1032	1296	2984	4393	5483	12643	19024
	5	176	326	423	683	1140	1537	2531	4170	5942
	10	96	197	262	429	750	936	1350	2412	3538
nb iter.	**1**	32	49	61	55	85	110	91	154	216
	5	47	70	86	83	117	148	124	210	269
	10	47	89	107	107	153	183	153	237	321
T. (s)	**1**	1	1	1	25	29	36	450	574	711
	5	2	2	2	38	39	50	567	755	889
	10	2	3	3	50	56	67	758	876	1136

* Le nombre cont. représente le nombre total de contraintes corrigées, toutes priorités confondues. Nb cpp. (resp. nb cmp.) est le nombre de contraintes les plus (resp. les moins) prioritaires qui ont été corrigées. T représente le temps d'exécution de l'algorithme.

TAB. 1 – Résultats expérimentaux obtenus lors de l'application de l'algorithme de fusion locale avec priorité sur des instances aléatoires de STPs avec priorité

6 Conclusion

Dans ce papier, nous nous sommes intéressés à la fusion de problèmes temporels simples STPs munies de priorités. Étant donné un ensemble de STPs avec priorité à fusionner, nous considérons le STP avec priorité S résultant de leur union. Si S est cohérent, alors la fusion est faite. Dans le cas contraire, la cohérence de S doit être rétablie. Nous avons présenté un principe de fusion avec priorité sur lequel est basé l'algorithme de fusion locale que nous avons proposé. Il existe deux motivations pour la stratégie locale : la première est la grande complexité de la détection exhaustive des conflits. La seconde est reliée à la nature des conflits. La détection puis la correction d'un ensemble de conflits éliminerait des conflits qui n'ont pas encore été détectés, ce qui a pour effet l'accélération de l'opération de fusion.

Dans le futur, nous espérons étendre ce travail au traitement des problèmes temporels disjonctifs.

Références

BENFERHAT S., DUBOIS D. & PRADE H. (1997). Some syntactic approaches to the handling

of inconsistent knowledge bases : A comparative study part 1 : The flat case. *Studia Logica*, **58**, 17–45.

BENFERHAT S., DUBOIS D., PRADE H. & WILLIAMS M. (2002). A practical approach to revising prioritized knowledge bases. *Studia Logica*, **70**(1), 105–130.

BENFERHAT S. & GARCIA L. (2002). Handling locally stratified inconsistent knowledge bases. *Studia Logica*, **70**, 77–104.

CORMEN T., LEISERSON C. & RIVEST R. (1990). *Introduction to Algorithms*. Cambridge, Massachusetts : MIT Press.

DECHTER R., MEIRI I. & PEARL J. (1991). Temporal constraint networks. *Artificial Intelligence*, **49**, 61–95.

GAREY M. & JOHNSON D. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman and co.

KHELFALLAH M. & BENHAMOU B. (2004a). Geographic information revision based on constraints. In *Proc. of the 14th European Conference on Artificial Intelligence, ECAI'04*, p. 828–832.

KHELFALLAH M. & BENHAMOU B. (2004b). Two revision methods based on constraints : Application to a flooding problem. In *Proc. of the 7th Int. Conf. of Artificial Intelligence and Symbolic Computation AISC'04*, volume 3249 of *LNAI*, p. 265–270.

KOLISCH R. & PADMAN R. (2001). An integrated survey of deterministic project scheduling. *Omega*, **29**, 249–272.

KONIECZNY S., LANG J. & MARQUIS P. (2002). Distance based merging : A general framework and some complexity results. In *Proc. of the 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'02)*, p. 97–108.

G. KUPER, G. L. LIBKIN & J. PARADAENS, Eds. (2000). *Constraint Databases*. Springer-Verlag.

LEISERSON C. & SAXE J. (1983). A mixed-integer linear programming problem which is efficiently solvable. In *Proc. of the 21st annual Allerton conference on Communications, Control, and Computing*, p. 204–213.

LIA Y. & WONG C. (1983). An algorithm to compact a vlsi symbolic layout with mixed constraints. In *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, volume 2, p. 62–69.

LIN J. & MENDELZON A. (1999). *Dynamic Worlds : From the Frame Problem to Knowledge Management*, volume 12 of *Applied Logic Series*, chapter Knowledge Base Merging by Majority. Kluwer.

MEYER T., GHOSE A. & CHOPRA S. (2001). Syntactic representation of semantic merging operations. In *Workshop on "Inconsistency in data and knowledge", in Int. Joint Conf. on Artificial Intelligence (IJCAI'01)*, p. 36–42.

RIGAUX P., SCHOLL M. & VOISARD A. (2002). *Spatial Databases with Application to GIS*. Morgan Kaufmann.

SHOSTAK R. (1981). Deciding linear inequalities by computing loop residues,. *Journal of ACM*, **28**(4), 769–779.

Apprentissage interactif de réseau de contraintes

Mathias Paulin

LIRMM - CNRS (UMR 5506)
161 rue Ada 34392 Montpellier cedex 5 - France
paulin@lirmm.fr

Abstract :

La programmation par contraintes est une technologie désormais largement utilisée pour résoudre des problèmes combinatoires dans les applications industrielles. Cependant, malgré le succès croissant qu'elle connaît, la diffusion de la programmation par contraintes est freinée par le manque d'experts connaissant ce paradigme. Depuis trois ans, une partie de la communauté "contraintes" travaille sur l'acquisition automatique de réseau de contraintes à partir d'instances que l'utilisateur accepte ou n'accepte pas comme solution à son problème. C'est dans cette optique que la plateforme Conacq a été proposée. Néanmoins, jusqu'à présent, cette dernière est passive vis à vis de l'utilisateur, c'est à dire basée sur la capacité de ce dernier à fournir des exemples significatifs de son problème. Dans cet article, nous proposons une amélioration de cette plateforme en développant un Conacq interactif, capable de poser à l'utilisateur des questions dont le but est d'augmenter plus rapidement et de manière conséquente la connaissance de la plateforme. Nous étudierons donc ici différentes approches de génération de questions dans le cadre de l'apprentissage de réseau de contraintes.

Mots-clés : Programmation par contraintes, Apprentissage.

1 Introduction

La programmation par contraintes (CP) connaît un succès croissant depuis qu'elle a montré sa capacité à résoudre des problèmes réels difficiles hors de portée des autres approches. Cependant, malgré ce succès, la diffusion de la CP est principalement freinée pour deux raisons. La première vient du fait que de nombreuses applications réelles (la mise en place des emplois du temps par exemple) qui pourraient être résolues à l'aide de réseaux de contraintes, ne font pas appel à la CP car les experts "métiers" (la personne chargée du planning dans notre exemple) en charge de ces applications ne maîtrisent pas la CP, et sont donc incapables modéliser leur problème sous la forme de réseaux de contraintes. Le second frein à la diffusion de la CP résulte du manque d'experts capables de modéliser efficacement des problèmes très combinatoires qui ne peuvent être

résolus à l'aide de modèles naïfs. Notre article n'a pas pour objectif de s'attaquer au second point, qui relève d'un problème de reformulation, mais présente des solutions capables d'aider un expert "métier", totalement novice en CP, à modéliser son problème sous la forme d'un réseau de contraintes.

Depuis trois ans, une partie de la communauté "contraintes" travaille sur l'acquisition automatique de réseau de contraintes à partir d'instances que l'utilisateur accepte ou n'accepte pas comme solution à son problème (Legtchenko Lallouet 2005). C'est dans cette optique que la plateforme CONACQ a été proposée (Coletta Bessière *et al.* 2003). Néanmoins, jusqu'à présent, cette dernière est passive vis à vis de l'utilisateur, c'est à dire basée sur la capacité de ce dernier à fournir des exemples significatifs de son problème. Dans cet article, nous proposons une approche théorique, validée par des expérimentations simples, permettant d'améliorer cette plateforme en développant un CONACQ interactif, capable de poser à l'utilisateur des questions dont le but est d'augmenter plus rapidement et de manière conséquente la connaissance acquise.

Le reste de cet article est organisé comme suit. La section 2 rappelle quelques définitions préliminaires sur les réseaux de contraintes et résume succinctement le fonctionnement de la plateforme CONACQ . La section 3 présente ensuite une première approche, basée sur les principes d'Active Learning (Cohn Ghahramani *et al.* 1996), consistant à effectuer une sélection judicieuse d'instances générées aléatoirement. Dans la section 4, nous caractérisons la question optimale et montrons comment la construire. La section 5 illustre les deux modules précédemment présentés au travers de quelques expérimentations préliminaires, avant de conclure le travail (section 6).

2 Préliminaires

Dans cette section, nous introduisons les concepts de base utilisés dans l'article. Nous rappelons quelques définitions de la programmation par contraintes avant de présenter succinctement la plateforme CONACQ ainsi que quelques propriétés de l'Espace des Versions.

2.1 Rappel sur la programmation par contraintes

Un réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est défini par un ensemble fini $\mathcal{X} = \{x_1, \ldots, x_n\}$ de n variables prenant chacune une valeur de leur domaine respectif $D_{x_1}, \ldots, D_{x_n}$, éléments de $\mathcal{D}$, et par $\mathcal{C} = \{C_1, \ldots, C_m\}$ une séquence de contraintes sur $\mathcal{X}$. Une contrainte C_i est définie par la séquence $var(C_i)$ des variables sur lesquelles elle porte, et par la relation $sol(C_i)$ qui spécifie les n-uplets autorisés sur $var(C_i)$. L'assignement de valeurs aux variables de $var(C_i)$ satisfait C_i si il appartient à $sol(C_i)$. Une instance e sur $\mathcal{X}$ est un n-uplet $(v_1, \ldots, v_n) \in D_{x_1} \times \ldots \times D_{x_n}$. On dit qu'une instance est une solution du réseau si elle satisfait toutes les contraintes du réseau. Sinon, c'est une non solution. On note $Sol(\mathcal{X}, \mathcal{D}, \mathcal{C})$ l'ensemble des solutions de $(\mathcal{X}, \mathcal{D}, \mathcal{C})$. Lorsque toutes les contraintes de $\mathcal{C}$ mettent en jeu exactement deux variables, nous disons que les contraintes et le réseau sont *binaires*. Dans cet article, nous restreignons notre étude au cas binaire ; La notation C_{ij} signifie dans ce cas que

la contrainte C est placée sur les variables x_i et x_j du problème.

L'apprentissage de réseau de contraintes a pour premier but de soulager l'utilisateur en fournissant à ce dernier des méthodes semi-automatiques pour acquérir les contraintes du problème qu'il cherche à modéliser. Comme point de départ, nous supposons que l'utilisateur connaît l'ensemble $\mathcal{X}$ des variables du problème ainsi que leur domaine $\mathcal{D}$ de valeurs possibles. Il est aussi supposé capable de se prononcer sur la validité d'une instance, il fournit à ce titre E^+ un sous-ensemble des solutions de son problème et E^- un ensemble de non solutions. Par ailleurs, l'objectif de CONACQ consiste à modéliser le problème de l'utilisateur dans un solveur de contraintes. Notre biais d'apprentissage, noté $\mathcal{B}$ est en conséquence une librairie de contraintes issue de ce solveur.

On dit qu'une séquence de contraintes appartient au biais si et seulement si cette séquence n'utilise que des contraintes de la librairie du biais.

Apprendre un réseau de contraintes consiste à rechercher une séquence de contraintes $\mathcal{C}$ appartenant à un biais d'apprentissage $\mathcal{B}$ donné, et dont l'ensemble des solutions est un sur-ensemble de E^+ ne contenant aucun élément de E^-. Le Problème d'Acquisition de Contraintes se définit alors formellement comme suit :

Définition 1 (Problème d'Acquisition de Contraintes)
Étant donnés un ensemble de variables $\mathcal{X}$, leur domaine $\mathcal{D}$, deux ensembles E^+ et E^- d'instances sur $\mathcal{X}$, et un biais d'apprentissage $\mathcal{B}$, le problème d'acquisition de contraintes consiste à trouver une séquence de contraintes $\mathcal{C}$ telle que :

$$\begin{cases} \mathcal{C} \in \mathcal{B}, \\ \forall e^- \in E^-, e^- \textit{ n'est pas solution de } (\mathcal{X}, \mathcal{D}, \mathcal{C}), \textit{ et,} \\ \forall e^+ \in E^+, e^+ \textit{ est solution de } (\mathcal{X}, \mathcal{D}, \mathcal{C}). \end{cases}$$

2.2 Processus d'apprentissage de la plateforme CONACQ

Présentée en 2003 (Coletta Bessière *et al.* 2003), CONACQ est une plateforme d'apprentissage de réseau de contraintes basée sur l'Espace des Versions (Mitchell 1997). Dans cet article, nous considérons CONACQ comme une boite noire qui, étant donnés deux ensembles E^+ et E^- de données d'entraînement,[1] renvoie une formule SAT, notée $\mathcal{K}$. Le vocabulaire de cette base est un ensemble d'atomes y^r_{ij}, où r est une contrainte de $\mathcal{B}$ et i et j sont tels que x_i et x_j appartiennent à $\mathcal{X}$. Le littéral y^r_{ij} signifie, lorsqu'il est satisfait, que la contrainte r doit être placée entre les variables x_i et x_j du problème. Un résultat notoire de la formulation SAT de CONACQ (Bessière Coletta *et al.* 2005) est : *Pour toute solution de $\mathcal{K}$, le réseau de contraintes associé est consistant avec les données d'entraînement.*

La construction de $\mathcal{K}$ suit le mode opératoire suivant. Pour chaque instance d'entraînement e fournie par l'utilisateur, CONACQ construit l'ensemble $\mathcal{K}_e$ des littéraux correspondant aux contraintes les plus générales qui permettent de rejeter e. La

[1] respectivement ensemble de solutions et de non solutions au problème de l'utilisateur.

mise à jour de la base de connaissance $\mathcal{K}$ diffère ensuite selon la classe d'appartenance de e :

- Si e est une instance négative, alors une seule des contraintes identifiées dans $\mathcal{K}_e$ suffit à expliquer le rejet de e. On ajoute donc à $\mathcal{K}$ la disjonction des éléments de $\mathcal{K}_e$.
- Si e est une instance positive, alors CONACQ est assurée qu'aucune des contraintes identifiées dans $\mathcal{K}_e$ n'appartient au concept cible (sinon, e serait une non solution). En conséquence, $\mathcal{K}$ est mis à jour en ajoutant la conjonction des négations des littéraux de $\mathcal{K}_e$.

L'analyse des instances positives de E^+ permet de simplifier les clauses de rejet des instances négatives de E^- et permet, *in fine*, d'identifier exactement quelles sont les contraintes mises en jeu dans le réseau cible.

L'exemple 1 permet d'illustrer simplement le mode de fonctionnement de la plateforme CONACQ .

Exemple 1

Dans cet exemple, nous cherchons un réseau de contraintes mettant en jeu trois variables x_1, x_2 et x_3 dont les domaines sont $D(x_i) = \{1, 2, 3, 4\}$. Notre biais d'apprentissage est restreint à $\mathcal{B} = ((\{x_1, x_2\}, L), (\{x_2, x_3\}, L))$ où L est la librairie de contraintes arithmétiques binaires $L = \{<, \leq, =, \geq, >, \neq\}$. Soient maintenant $E^- = \{(3, 2, 1), (3, 3, 2)\}$ et $E^+ = \{(1, 3, 2)\}$ les données d'entraînement.

Comme le montre le tableau 1, l'analyse des instances négatives implique une mise à jour de $\mathcal{K}$ au moyen de clauses disjonctives alors que les instances positives permettent d'augmenter $\mathcal{K}$ par conjonction de littéraux mis à Faux.

instance	$\mathcal{K}_e$	$\mathcal{K}$
$e_1^- = (3, 2, 1)$	$\{y_{12}^{\leq}; y_{23}^{\leq}\}$	$(y_{12}^{\leq} \vee y_{23}^{\leq})$
$e_2^- = (3, 3, 2)$	$\{y_{12}^{\neq}; y_{23}^{\leq}\}$	$(y_{12}^{\leq} \vee y_{23}^{\leq}) \wedge (y_{12}^{\neq} \vee y_{23}^{\leq})$
$e_1^+ = (1, 3, 2)$	$\{y_{12}^{>}; y_{23}^{<}\}$	$(y_{12}^{\leq}) \wedge (y_{12}^{\neq}) \wedge (\neg y_{12}^{>}) \wedge (\neg y_{23}^{<})$

Table 1: Évolution de la base de connaissance de CONACQ pour l'exemple 1.

Après l'analyse de e_1^-, $(y_{12}^{\leq} = 1)$ est une solution pour $\mathcal{K}$. $P_1 = \{(x_1 \leq x_2, x_2 = x_3)\}$ est alors un réseau consistant. Une solution pour $\mathcal{K}$ après e_2^- peut être $(y_{12}^{\leq} = y_{12}^{\neq} = 1)$, on extirpe alors $P_2 = \{(x_1 < x_2, x_2 = x_3)\}$ réseau de contraintes consistant. Le réseau $P_3 = \{(x_1 < x_2, x_2 \geq x_3)\}$ est enfin consistant avec $E = E^+ \cup E^-$ après analyse de e_1^+.

2.3 Espace des Versions

Comme nous l'avons précisé précédemment, CONACQ est basée sur l'Espace des Versions (Mitchell 1997), un paradigme d'apprentissage automatique présentant de

bonnes propriétés (incrémentalité, commutativité, relativement à l'ordre des données d'entraînement).

Définition 2 (Espace des Versions)
Étant donné $(\mathcal{X}, \mathcal{D})$ un ensemble de variables et leur domaine, E^+ et E^- deux ensembles d'entraînement et H un ensemble d'hypothèses, l'espace des versions est l'ensemble :

$$V = \{h \in H | E^+ \subseteq Sol(\mathcal{X}, \mathcal{D}, h), E^- \cap Sol(\mathcal{X}, \mathcal{D}, h) = \emptyset\}$$

Grâce à un ordre partiel $\leq_G$, un espace des versions est complètement caractérisé par deux bornes : la borne spécifique S qui est l'ensemble des éléments minimaux de V (selon $\leq_G$), et la borne générale G, celui des éléments maximaux.

Définition 3
Étant donné un ensemble E d'instances, on note S_{ij}^E l'état de la borne spécifique S relative au couple (x_i, x_j) après l'analyse de E.

Dans le cadre de l'apprentissage de réseau de contraintes, une hypothèse h est une séquence de contraintes appartenant à $\mathcal{B}$ et l'Espace des Versions est l'ensemble de tous les réseaux de contraintes consistants du problème d'acquisition de contraintes. Par ailleurs, la propriété des solutions la formule $\mathcal{K}$ produite par la plateforme CONACQ (*c.f.* 2.2) et la définition 2 permettent de déduire que l'ensemble $Sol(\mathcal{K})$ des solutions de la base $\mathcal{K}$ matche exactement l'Espace des Versions après analyse des données d'entrainement.

3 Une première approche de questionnement

À l'heure actuelle, CONACQ attend passivement les instances fournies par l'utilisateur. Cette approche est cependant perfectible, dans la mesure où elle repose sur la capacité pour l'utilisateur de fournir des instances significatives de son problème. Dans cette section, nous présentons une première technique de questionnement basée sur la sélection judicieuse d'instances générées aléatoirement.

3.1 Principe général

L'attente passive de CONACQ vis à vis des données d'entraînement constitue pour le moment un défaut majeur pour la plateforme. En effet, cela suppose de la part de l'utilisateur la capacité de produire des exemples significatifs de son problème. En pratique, le manque d'efficacité du processus d'apprentissage résulte de deux causes principales :

1. L'utilisateur est limité par ses propres connaissances du problème.[2] Les exemples qu'il fournit sont en général relativement similaires (peu de valeurs diffèrent

[2]Car le problème est difficile à résoudre ; c'est d'ailleurs pour cette raison que l'utilisateur souhaite utiliser CONACQ .

d'une instance à l'autre). Dans ce cas leur analyse augmente peu, voire pas, la connaissance acquise.

2. Durant le processus d'apprentissage, deux instances apparemment très différentes peuvent être sémantiquement identiques pour un biais d'apprentissage donné. Par exemple, $e_1 = (1, 2, 3, 4)$ et $e_2 = (2, 3, 4, 5)$ produisent la même connaissance si on utilise la librairie arithmétique binaire $L = \{<, \leq, =, \geq, >, \neq\}$.

Pour éviter ces problèmes, nous proposons un premier générateur de questions. Notre approche consiste à doter la plateforme d'un générateur d'instances aléatoires que CONACQ pourra utiliser de la manière suivante : il présentera à l'utilisateur des instances e générées aléatoirement afin que ce dernier statue sur leur validité. La réponse obtenue servira alors à augmenter la connaissance acquise. Doté d'un tel générateur, la plateforme pourrait alors combattre les problèmes liés au biais de l'utilisateur (point 1).

Pour combattre les problèmes de similarité sémantique (point 2), nous ajoutons la fonctionnalité suivante : Avant de demander à l'utilisateur de statuer sur la validité des instances générées aléatoirement, nous les présentons d'abord à CONACQ afin que ce dernier réalise une sélection parmi ces instances. Pour chaque instance e générée, CONACQ estime le gain de connaissance qu'apporterait e à la connaissance actuelle $\mathcal{K}$. e n'est alors présentée que dans le cas où la réponse de l'utilisateur apporte un gain de connaissance. Avec cette approche, nous mettons en place une sélection des questions dans le but d'améliorer sans cesse la connaissance acquise. Notre démarche s'inscrit donc dans le cadre de l'Active Learning décrit dans (Cohn Ghahramani *et al.* 1996).

En utilisant l'Espace des Versions (c.f. section 2.3), à chaque étape du processus d'apprentissage, la propriété suivante est vérifiée : Chaque instance rejetée par une contrainte de la borne générale G de l'espace des versions est rejetée par tous les réseaux de contraintes encore possibles. De même, toute instance acceptée par la borne spécifique S est acceptée par tous les réseaux de contraintes encore disponibles. Enfin, toute instance non acceptée par S mais non rejetée par G est seulement acceptée par une partie des réseaux de contraintes encore disponibles (et rejetée par les autres).

Nous déduisons de cette propriété que les instances intéressantes pour CONACQ sont celles qui sont situées entre les bornes S et G de l'Espace des Versions.

Illustrons maintenant le principe de sélection à l'aide de l'exemple 2.

Exemple 2
Dans cet exemple, nous cherchons un réseau de contraintes mettant en jeu trois variables $\mathcal{X} = \{x_1, x_2, x_3\}$ où $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3, 4, 5, 6\}$. Nous utilisons pour cela le biais d'apprentissage $\mathcal{B} = ((\{x_1, x_2\}, L), (\{x_2, x_3\}, L))$, où L est la librairie $\{<, \leq, =, \geq, >, \neq\}$. Soient $e_1^+ = (2, 2, 5)$ et $e_2^- = (1, 3, 2)$ les deux premières instances d'entraînement. Après les avoir analysées, le réseau de contraintes le plus spécifique construit met en jeu deux contraintes $P_S = \{(x_1 = x_2, x_2 < x_3)\}$. Il existe cependant deux possibilités pour le rejet de e_2^-, le réseau de contraintes issu de la borne générale G s'exprime donc de la façon suivante : $P_G = \{(x_1 \geq x_2), (x_2 \leq x_3)\}$.

Soient maintenant $i_1 = (4, 4, 6)$, $i_2 = (5, 6, 1)$ et $i_3 = (5, 5, 5)$ trois instances produites par notre générateur aléatoire. La première instance i_1 ne sera pas présentée[3] à l'utilisateur car elle est déjà acceptée par le réseau le plus spécifique P_S. Il est tout autant inutile de demander à l'utilisateur de statuer sur la validité de i_2 car cette dernière est déjà rejetée par P_G. Enfin, $i_3 = (5, 5, 5)$ constitue une instance intéressante qui doit être présentée à l'utilisateur, dans la mesure où la connaissance actuelle ne permet pas de statuer sur sa validité. Si l'utilisateur considère que i_3 est une solution à son problème, la borne spécifique S sera mise à jour $P_{S'} = \{(x_1 = x_2, x_2 \leq x_3)\}$. Sinon ($i_3$ est une non solution), la borne générale G sera abaissée à $P_{G'} = \{(x_1 \geq x_2, x_2 \leq x_3), (x_2 < x_3)\}$.

3.2 Formalisation

Dans cette section, nous présentons une formalisation du processus de sélection des instances générées aléatoirement. Soit e_q une instance ainsi générée et soit Cl_q la clause codant pour son rejet, telle que Cl_q a été simplifiée par propagation unitaire de la connaissance $\mathcal{K}$. Si $Cl_q = \emptyset$, alors e_q est dores et déjà acceptée par S, elle sera donc filtrée par le module de sélection, c'est à dire qu'elle ne sera pas présentée à l'utilisateur. Si il existe une clause $Cl \in \mathcal{K}$ telle que $Cl_q \supseteq Cl$, alors e_q est déjà rejetée par la borne spécifique G et sera là encore filtrée. Toutes les autres instances générées sont intéressantes en vue d'une augmentation de la connaissance de CONACQ .

Il convient cependant de noter que deux cas se distinguent plus particulièrement : $|Cl_q| = 1$ et $Cl_q \subset Cl$. La méthode du *near-miss* présentée dans (Smith Rosenbloom 1990) traite du cas où $|Cl_q| = 1$. Dans le cas où $Cl_q \subset Cl$, la mise à jour de Cl dépend de la classification de e_q par l'utilisateur. Si e_q est déclarée négative, Cl prend pour nouvelle valeur Cl_q. Dans le cas contraire, Cl devient $Cl \backslash Clq$.

Exemple 3
Dans cet exemple, et à l'aide du tableau 2, nous illustrons nos précédents propos en décrivant le processus de sélection effectuée par CONACQ lors de l'analyse des instances de l'exemple 2. Il convient de remarquer que Cl_q se vide par propagation unitaire de $\mathcal{K}$ pour i_1, et $Cl_q \supseteq Cl$ pour i_2. Enfin, la clause Cl_q construite pour l'analyse de la troisième instance permet de conclure que i_3 est une question intéressante qui peut être présentée à l'utilisateur.

Pour terminer cette section, il convient de noter que la taille de la clause $|Cl_q|$ correspond à la distance qui sépare e_q de la borne spécifique S. En conséquence, si $|Cl_q|$ est grande alors e_q est très proche de la borne générale dans l'Espace des Versions. Dans ce cas, le gain de connaissance pour CONACQ sera très important si e_q est une solution au problème de l'utilisateur. En revanche, le gain sera réduit si e_q est une non solution. Symétriquement, si $|Cl_q|$ est réduite, l'apport de connaissance sera important si e_q est une instance négative du problème, mais réduit dans le cas contraire. En se basant sur la taille des clauses Cl_q, nous pouvons envisager une sélection des instances aléatoires

[3]Sauf si nous cherchons un effondrement de l'Espace de Versions afin de montrer que le biais d'apprentissage $\mathcal{B}$ n'est pas assez expressif pour le problème étudié.

Instance	Analyse de CONACQ	Cl_q resolved by $\mathcal{K}$
$e_1^+ = (2,2,5)$	$\mathcal{K} = (\neg y_{12}^{\neq}) \wedge (\neg y_{23}^{>})$	–
$e_2^- = (1,3,2)$	$Cl = (y_{12}^{>} \vee y_{23}^{<})$	–
$i_1 = (4,4,6)$	$Cl_q = (y_{12}^{\neq} \vee y_{23}^{>})$	$\emptyset$
$i_2 = (5,6,1)$	$Cl_q = (y_{12}^{>} \vee y_{23}^{<})$	$(y_{12}^{>} \vee y_{23}^{<}) = Cl$
$i_3 = (5,5,5)$	$Cl_q = (y_{12}^{\neq} \vee y_{23}^{\neq})$	$(y_{23}^{\neq})$

Table 2: Sélection effectuée par CONACQ pour les instances de l'exemple 2.

selon leur positionnement dans l'espace de versions ; nous sommes dans ce cas dans le cadre d'une généralisation du *near-miss*. Il convient par ailleurs de noter que de telles heuristiques ont été étudiées (O'Connell O'Sullivan *et al.* 2002) dans le cadre de l'apprentissage d'une contrainte unique.

4 Optimalité du questionnement

La section précédente a permis de dégager une première approche de questionnement. Cependant, le gain de connaissance apporté par une question varie selon la position de cette dernière dans l'Espace des Versions et selon la réponse donnée par l'utilisateur. Dans notre recherche d'optimisation, il convient maintenant d'identifier la meilleure question à poser au cours du processus d'apprentissage, c'est à dire celle dont le gain de connaissance est maximum quelque soit la réponse de l'utilisateur.

Notre objectif dans cette section consiste donc à caractériser une question optimale ainsi qu'à évaluer la complexité relative à sa construction.

4.1 Caractérisation d'une question optimale

À chaque étape du processus d'apprentissage, la base SAT $\mathcal{K}$ renferme l'ensemble des connaissances acquises. Soit $Cl = (y_{ij}^{r} \vee \ldots \vee y_{i'j'}^{r'})$ une clause de $\mathcal{K}$ codant le rejet d'une instance négative e (on suppose ici que Cl a été réduite par propagation unitaire). Cl code l'ensemble des contraintes qui rejètent e, et dont au moins une doit faire partie du réseau cible si nous voulons que ce dernier soit consistant avec les données d'entraînement.

L'objectif de l'interaction avec l'utilisateur consiste à trouver, parmi toutes ces contraintes, celle qui appartient réellement au réseau cible. Si nous utilisons la technique du *near-miss* (Smith Rosenbloom 1990), nous devons poser $|Cl|$ questions dans le pire des cas. Nous proposons ici une autre technique nous permettant de réduire le nombre d'interactions à $log(|Cl|)$ questions. Il convient de noter que notre idée est inspirée de celle de Mitchell (Mitchell 1997), qui garantit un nombre logarithmique de questions pour converger.

Propriété 1 (Caractérisation d'une question optimale)
Étant donnée $\mathcal{K}$ une base SAT représentant un ensemble de connaissances et $Cl \in \mathcal{K}$ une clause codant une non solution des données d'entraînement. Une instance e_q constitue une question optimale relativement à Cl si la clause Cl_q codant le rejet de e_q est telle que :

$$Cl_q \subset Cl \quad \textit{et} \quad |Cl_q| = \frac{|Cl|}{2}$$

Preuve. Soit une instance e_q telle que la clause Cl_q codant pour son rejet soit telle que $Cl_q \subset Cl$ et $|Cl_q| = |Cl|/2$. Si $e_q \in E^-$, alors Cl_q sera ajoutée à la base SAT $\mathcal{K}$. Comme $Cl_q \subset Cl$, Cl sera subsumée par Cl_q. Dans le cas contraire ($e_q \in E^+$), les littéraux y_{ij}^r appartenant à Cl_q seront négatés dans $\mathcal{K}$, réduisant ainsi, par propagation unitaire, Cl à $Cl \setminus Cl_q$. Quelque soit la classe d'appartenance de e_q (déterminée par l'utilisateur), e_q nous assure une division par 2 de la taille de Cl. En conséquence, e_q correspond bien à une question optimale puisque le gain de connaissance obtenu ne dépend pas de la réponse de l'utilisateur. □

Exemple 4
Pour illustrer nos propos, nous considérons dans cet exemple un ensemble de variables $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$ avec $D(x_i) = \{1, 2, 3, 4\} \forall i \in [1..5]$. Le biais d'apprentissage $\mathcal{B} = (\{\{x_i, x_{i+1}\}, L) \forall i \in [1..4]\})$, où $L = \{<, \leq, =, \geq, >, \neq\}$.

instance	CONACQ 's output
$e_1^+ = (1, 2, 4, 3, 1)$	$\mathcal{K} = (\neg y_{12}^{\geq}) \wedge (\neg y_{23}^{\geq}) \wedge (\neg y_{34}^{\leq}) \wedge (\neg y_{45}^{\leq})$
$e_2^- = (2, 2, 2, 2, 2)$	$Cl = (y_{12}^{\neq} \vee y_{23}^{\neq} \vee y_{34}^{\neq} \vee y_{45}^{\neq})$

Table 3: Analyse des instances pour l'exemple 4.

Le tableau 3 renferme les deux premières instances fournies par l'utilisateur ainsi que l'analyse correspondante effectuée par CONACQ . Après l'analyse de ces deux instances, le réseau de contraintes le plus spécifique établi par CONACQ est $P_S = (x_1 < x_2, x_2 < x_3, x_3 > x_4, x_4 > x_5)$ et l'ensemble des réseaux de contraintes permettant d'expliquer le rejet de e_2^- est $P_G = \{(x_1 \neq x_2), (x_2 \neq x_3), (x_3 \neq x_4), (x_4 \neq x_5)\}$.

Notre objectif est donc maintenant de déterminer quelle est la cause réelle du rejet de e_2^- en utilisant le procédé décrit dans nos propos précédents. La clause correspondant à e_2^- est $Cl = (y_{12}^{\neq} \vee y_{23}^{\neq} \vee y_{34}^{\neq} \vee y_{45}^{\neq})$. Nous allons dans un premier temps chercher à savoir si $(x_1 \neq x_2)$ et $(x_2 \neq x_3)$ appartiennent au réseau cible. La question optimale correspond en conséquence à $Cl_q = (y_{12}^{\neq} \vee y_{23}^{\neq})$.

Parmi nos hypothèses de travail, nous partons du principe que l'utilisateur ne maîtrise pas les paradigmes de la programmation par contraintes. [4] En accord avec le vocabulaire

[4] C'est d'ailleurs pour cela qu'il utilise la plateforme pour modéliser son problème sous la forme d'un réseau de contraintes.

défini par D. Angluin (Angluin 2004), nous supposons par conséquent que l'utilisateur n'est pas capable de répondre à des questions d'**équivalence** : *"Ce réseau de contraintes est-il équivalent au réseau recherché?"*. Nous supposons seulement qu'il est capable de répondre à des questions d'**appartenance**, c'est à dire des questions de type : *"Cette instance est-elle une solution à votre problème?"*

Forts de cette constatation, formuler une question optimale revient à extirper une instance e_q dont Cl_q serait la clause codant son éventuel rejet. Pour cela, il faut déduire de Cl_q un réseau de contraintes P, résoudre ce dernier, prendre e_q parmi l'ensemble des solutions de P, puis enfin présenter à l'utilisateur la question d'appartenance *"e_q est-elle une solution à votre problème?"*.

4.2 Construction d'une question optimale

Dans cette partie, à l'aide de l'algorithme 1, nous présentons une formalisation du processus permettant de créer une question optimale à partir d'une clause Cl_q.

Algorithm 1: Construction d'une question optimale

Entrée: $Cl_q, E, \mathcal{X}, \mathcal{D}$
Sortie: e_q : question optimale relativement à Cl_q

$C \leftarrow \emptyset$
$Cl \leftarrow$ clause dont est issue Cl_q
$e \leftarrow$ instance dont Cl code le rejet
pour $(x_i, x_j) \in (\mathcal{X})^2$ **faire**
1 **si** (i,j) *non mis en jeu dans* Cl_q **alors** $C_{ij} \leftarrow S_{ij}^{E}$
2 **si** (i,j) *mis en jeu dans* Cl_q **alors** $C_{ij} \leftarrow S_{ij}^{\{e\}}$
$C \leftarrow C \cup C_{ij}$
$P \leftarrow (\mathcal{X}, \mathcal{D}, C)$
3 $Sol \leftarrow$`résoudre`(P)
retourner *un élément de* Sol

Le fonctionnement de l'algorithme 1 est le suivant : Si Cl est une clause de $\mathcal{K}$ codant une instance négative $e \in E^-$. Soit Cl_q une clause issue de Cl correspondant aux spécifications données dans la section précédente. Nous cherchons à produire un réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, C)$ dont l'ensemble $Sol(\mathcal{X}, \mathcal{D}, C)$ des solutions est tel que $\forall e_q \in Sol(\mathcal{X}, \mathcal{D}, C)$, si e_q est rejetée par l'utilisateur, alors la clause codant son rejet correspond à la clause Cl_q.

Nous voulons pour cela que seules les contraintes codées par les littéraux de Cl_q soient potentiellement violées. En conséquence, il nous faut placer chaque contrainte C_{ij} non impliquée dans Cl_q à sa borne spécifique dans l'Espace des Versions. Nous avons donc $rel(C_{ij}) = S_{ij}^{E}$ (ligne 1). De façon symétrique, nous cherchons à ce que

toutes les contraintes impliquées dans Cl_q soient potentiellement violées. Nous forçons en conséquence (ligne 2) $rel(C_{ij}) = S_{ij}^{\{e\}}$ (i.e. borne spécifique de l'Espace des Versions lorsque e est la seule instance étudiée).

Si l'on note maintenant $C = (C_{ij}, \ldots, C_{i'j'})$ la séquence des contraintes C_{ij} ainsi identifiées, il suffit de lancer (ligne 3) une résolution de $P = (\mathcal{X}, \mathcal{D}, C)$, puis de présenter à l'utilisateur une instance $e_q \in Sol(\mathcal{X}, \mathcal{D}, C)$.

Propriété 2 (Complétude et correction de l'algorithme 1)
L'algorithme 1 est correct et complet : L'instance e_q qu'il calcule et renvoie correspond à la question optimale relativement aux données d'entrée.

Preuve. Soit Cl la clause codant pour une instance négative e de E, Cl_q la clause codant une question optimale et respectant la propriété 1, et e_q l'instance renvoyée par l'algorithme 1. Supposons que e_q soit une instance négative du problème, et notons Cl_{e_q} la clause codant son rejet. Par définition du réseau de contraintes P construit, e_q ne viole aucune des contraintes portant sur les couples (i, j) n'étant pas mis en jeu dans Cl_q (puisque ces contraintes on été placées à S_{ij}^E). Étudions maintenant les contraintes C_{ij} posées sur les couples mis en jeu dans Cl_q. Pour chacune d'elle, $C_{ij} = S_{ij}^{\{e\}}$, le tuple (v_i, v_j) est par conséquent sémantiquement équivalent au couple (v'_i, v'_j) de e. Les littéraux y_{ij}^r de Cl_{e_q} sont donc les mêmes que ceux de Cl_q, ce qui achève de démontrer que $Cl_{e_q} = Cl_q$. □

Enfin, il convient de noter que l'algorithme 1 nécessite la résolution d'un réseau de contraintes (ligne 3) afin d'en extirper une solution. La construction d'une question optimale se révèle donc être un procédé NP-complet, contrairement au cadre de l'attribut/valeur décrit par Mitchell.

Exemple 5
Afin d'illustrer le processus de construction présenté ci-dessus, nous reprenons l'étude de l'exemple 4 afin d'extirper la question optimale correspondant à la clause $Cl_q = (y_{12}^{\neq} \vee y_{23}^{\neq})$.

(x_1, x_2) et (x_2, x_3) sont les seuls couples mis en jeu dans Cl_q. En conséquence, les contraintes relatives aux autres couples sont déterminées à l'aide de $S_{34}^E = S_{45}^E = " > "$. $(x_3 > x_4)$ et $(x_4 > x_5)$ sont donc ajoutées au réseau P en cours de construction.

Nous cherchons ensuite à ce que seules les contraintes $(x_1 \neq x_2)$ et $(x_2 \neq x_3)$ soient potentiellement violées. Nous utilisons pour cela $S_{12}^{\{e\}} = S_{23}^{\{e\}} = " = "$. Nous obtenons donc finalement le réseau de contraintes $P = \{(x_1 = x_2, x_2 = x_3, x_3 > x_4, x_4 > x_5)\}$.

L'instance $e_q = (3, 3, 3, 2, 1)$ est une solution du réseau de contraintes ainsi construit. Il ne reste plus qu'à la présenter à l'utilisateur. Si ce dernier la classifie négativement, nous serons assurés qu'au moins une des deux contraintes $(x_1 \neq x_2)$ ou $(x_2 \neq x_3)$ appartient au réseau cible. Dans le cas contraire ($e_q \in E^+$), nous serons assurés que ces deux contraintes n'appartiennent pas au réseau cible.

5 Expérimentations

Afin de comparer l'impact des deux approches que nous avons proposées dans cet article, nous avons mis en place une plateforme expérimentale d'apprentissage interactif de contraintes. Pour implémenter cette dernière, notre choix s'est porté sur le langage JAVA, nous permettant ainsi d'utiliser le solveur CHOCO (Choco) pour les aspects contraintes et la librairie SAT4J (Sat4J) pour les aspects *satisfaction de formules booléennes*.

Lors de nos expérimentations, nous avons utilisé la librairie de contraintes $L = \{<, \leq, =, \geq, >, \neq\}$ et généré aléatoirement des réseaux de contraintes cibles mettant en jeu uniquement des contraintes de L. Le tableau 4 recense les résultats obtenus lors de nos expérimentations, et durant lesquelles nous avons fait varier le nombre $|\mathcal{X}|$ de variables du problème, le nombre $|\mathcal{C}|$ de contraintes appartenant au réseau cible ainsi que le nombre $\#q$ d'instances ou questions posées puis analysées par CONACQ . Les trois autres colonnes présentent les résultats obtenus après $\#q$ interactions pour les différentes techniques d'apprentissage présentées dans cet article : CONACQ *standard*, la sélection des questions et enfin l'apprentissage par questions optimales. $\#y_{ij}^r$ représente le nombre de littéraux de la base $\mathcal{K}$ non encore assignés à la fin du processus d'apprentissage, $\#sol$ le nombre de solutions pour $\mathcal{K}$, et $t(q)$ le temps moyen nécessaire à la génération d'une instance.

Le nombre $\#sol$ de solutions de la base $\mathcal{K}$ correspond au nombre de réseaux consistants avec les données d'entraînement, et donc à la taille de l'Espace des Versions. Nous pouvons par conséquent considérer que plus $\#sol$ est élevé, moins la connaissance acquise par CONACQ est précise. Par ailleurs, le nombre $\#y_{ij}^r$ de littéraux non assignés nous permet d'obtenir une borne sur le nombre de solutions : $\#sol \leq 2^{\#y_{ij}^r}$. Cette donnée est par conséquent intéressante lorsque le temps de calcul du nombre de solutions devient trop important.

Paramètres			CONACQ *standard*			*Sélection des questions*			*Questionnement optimal*		
$\|\mathcal{X}\|$	$\|\mathcal{C}\|$	$\#q$	$\#y_{ij}^r$	$\#sol$	$t(q)$	$\#y_{ij}^r$	$\#sol$	$t(q)$	$\#y_{ij}^r$	$\#sol$	$t(q)$
3	1	5	4	12	$0.10s$	3	6	$0.12s$	2	4	$0.16s$
3	2	10	6	21	$0.10s$	2	3	$0.15s$	1	2	$0.21s$
3	3	10	8	35	$0.10s$	6	24	$0.19s$	1	2	$0.32s$
5	6	10	30	–	$0.10s$	18	–	$0.33s$	4	12	$0.57s$
5	6	20	26	–	$0.10s$	7	28	$0.47s$	2	4	$0.89s$
8	10	20	62	–	$0.10s$	46	–	$0.58s$	29	–	$1.09s$
8	10	60	34	–	$0.10s$	28	–	$0.76s$	12	–	$1.72s$

Table 4: Comparaison entre les performances de CONACQ *standard*, la méthode de sélection des questions et l'apprentissage par questionnement optimal.

Les résultats de ces expérimentations nous permettent de formuler une première observation : La sélection des questions permet toujours d'obtenir des meilleurs résultats que l'apprentissage réalisé par CONACQ *standard*. De même, la connaissance acquise

par questionnement optimal est toujours plus précise que celle acquise par sélection des questions.

Nous notons par ailleurs que cette amélioration de connaissance a un impact sur le temps nécessaire à la génération d'une question : Le questionnement optimal est en effet plus lent que la méthode de sélection des questions, elle même plus lente que CONACQ *standard*. De plus, lorsque la taille du problème grandit, le temps nécessaire à CONACQ pour produire des instances aléatoires reste constant. Celui nécessaire à la sélection des questions semble quant à lui augmenter de manière linéaire, tandis que le temps de construction des questions optimales semble augmenter avec un facteur exponentiel qui pourrait s'expliquer par l'appel caché à la résolution de problèmes NP-complets.

Nous notons aussi que, pour une taille de problème donnée, le temps de génération d'une question augmente avec le nombre d'interactions, qui peut s'expliquer par le fait que plus la connaissance acquise s'affine, plus il est difficile de la consolider, et donc de générer une instance qu'on ne sait pas déjà classifier.

6 Conclusion et perspectives

Dans cet article, nous avons adapté au paradigme de l'apprentissage de réseau de contraintes les techniques d'Active Learning et d'apprentissage interactif par questions d'appartenance. Nous avons aussi dégagé une première méthode de questionnement basée sur la sélection judicieuse d'instances générées automatiquement, une approche facile à mettre en oeuvre. Cependant, notre étude montre qu'on ne maîtrise que partiellement le gain de connaissance fourni par la réponse à cette question. Nous avons ensuite caractérisé la stratégie d'interactions optimale et montré sa complexité théorique. Enfin, des résultats expérimentaux préliminaires ont permis de valider l'intérêt de ces deux approches.

En vue d'une utilisation sur des applications réelles, notre étude va continuer en étendant notre approche à d'autres biais d'apprentissage constitués de contraintes n-aires. Nous proposerons par ailleurs des alternatives à notre approche dans le cas où l'utilisateur n'arrive pas à se prononcer sur la validité d'une instance, par exemple lorsque cette dernière sera trop complexe à analyser.

References

D. Angluin *Queries revisited* Theoretical Computer Science, Volume 313, pages 175-194, Février 2004.

C. Bessière, R. Coletta, F. Koriche, B. O'Sullivan *A SAT-Based formulation of Conacq* Note Coconut 119, LIRMM – Université Montpellier II, Février 2005.

D. A. Cohn, Z. Ghahramani, M. I. Jordan *Active Learning with Statistical Models.* Journal of Artificial Intelligence Research 4, pages 129-145, Mars 1996.

R. Coletta, C. Bessière, B. O'Sullivan, E.C. Freuder, S. O'Connell, J. Quinqueton *Semi-automatic modeling by constraint acquisition.* Proceedings of CP-2003, Short paper, LNCS 2833, Springer Kinsale, Cork, Ireland., pages 812-816, Septembre 2003.

CHOCO *http://choco.sourceforge.net/* The Choco constraint programming system.

A. Legtchenko, A. Lallouet *Acquisition des contraintes ouvertes par apprentissage de solveurs.* To appear in CAP'05, Juin 2005.
T. Mitchell *Concept learning and the general-to-specific ordering.* Machine Learning, chapitre 2, pages 20-51. McGraw Hill, 1997.
SAT4J *http://www.sat4j.org/* A satisfiability library for Java.
B.D. Smith et P.S. Rosenbloom *Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces.* National Conference on Artificial Intelligence (AAAI-90), pages 848-853, 1990.
S. O'Connell, B. O'Sullivan, E.C. Freuder *Strategies for interactive constraint acquisition.* Proceedings of CP-2002 Workshop on User-Interaction in Constraint Satisfaction, pages 62-76, Septembre 2002.

Étude des symétries dans les réseaux de contraintes de différence

Mohamed Réda Saïdi, Belaïd Benhamou

Laboratoire des Sciences de l'Information et des Systèmes (LSIS)
Centre de Mathématiques et d'Informatique
39, rue Joliot Curie - 13453 Marseille cedex 13, France.
{saidi,Belaid.Benhamou}@cmi.univ-mrs.fr

Résumé :
La détection des valeurs symétriques lors de la résolution des CSPs offre la possibilité de réduire l'espace de recherche. Ceci s'explique par le fait que les valeurs symétriques à une valeur donnée sont dans un sens redondantes et leur suppression simplifie le problème sans pour autant modifier sa consistance. La vérification des conditions de symétrie est dans le cas général une tâche difficile à réaliser, cependant, dans le cas des CSPs à contraintes de différence ces conditions peuvent être simplifiées. Dans cet article nous montrons comment certaines valeurs sont éliminées lors de l'instanciation de la variable courante à une valeur menant à un échec. Nous donnons une condition suffisante prouvant que ces valeurs éliminées sont symétriques à la valeur de la variable courante. Nous proposons un algorithme de détection des valeurs symétriques de complexité linéaire dans la taille du problème. Ces symétries sont exploitées dans un algorithme de résolution de CSPs à contraintes de différence que nous expérimentons sur le problème de coloriabilité de graphes. Les résultats obtenus sont satisfaisants et montre un grand apport des symétries dans la preuve de consistance.

Mots-clés : Contraintes, problèmes de satisfaction de contraintes(CSP) , contraintes de différence, symétrie.

1 Introduction

Les problèmes de satisfaction de contraintes (CSPs) sont au coeur de nombreuses applications en intelligence artificielle et en recherche opérationnelle. On peut citer par exemple la visualisation de scènes en CAO, les problèmes d'ordonnancement, les systèmes experts, la coloration de graphes et tous les problèmes formalisés à l'aide de contraintes de manière générale.

Un CSP est défini comme un ensemble de contraintes impliquant des sous ensembles de variables. L'objectif étant de trouver une instanciation pour chaque variable satisfaisant l'ensemble des contraintes. Les CSPs font partie de la classe des problèmes NP-complets, les algorithmes complets connus pour les résoudre nécessitent un temps exponentiel en fonction de la taille du problème.

Dans la pratique plusieurs techniques efficaces ont été proposées, on peut citer la k-consistance (Freuder, 1978), la décomposition des problèmes (Jégou, 1990; Jégou, 1993; Dechter & Pearl, 1987; Dechter & Pearl, 1989) et l'élimination des symétries (Freuder, 1991; Benhamou, 1994; Crawford *et al.*, 1996; Focacci & Milano, 2001; Fahle *et al.*, 2001; Puget, 2001). Dans cet article nous nous intéressons au cas des CSPs *binaires à contraintes de différence*. Ce formalisme est suffisamment expressif pour représenter un grand nombre de problèmes du domaine de l'intelligence artificielle (planification, allocation de registre en compilation, cartographie,...). La détection des valeurs symétriques d'un domaine d'une variable d'un CSP quelconque est exponentielle dans le pire des cas (Benhamou, 1994). Par contre la détection de ces symétries dans le cas des CSPs à contraintes de différence est linéaire dans la taille du problème (Benhamou, 2004).

Nous montrons dans le cas où l'on s'intéresse qu'au problème de la consistance des CSPs à contraintes de différence, qu'il est possible de simplifier les conditions de la symétrie et en conséquence améliorer l'algorithme de détection de ces symétries. Nous montrons aussi comment cet affaiblissement des conditions de symétries peut faire émerger des nouvelles symétries non détectées dans (Benhamou, 2004).

Cet article est organisé comme suit : Nous rappelons un certain nombre de notions sur les CSPs dans la section 2. Dans la section 3 nous discutons de la notion de symétrie et montrons comment : (1)- affaiblir la condition de symétrie donnée dans (Benhamou, 2004) afin de favoriser l'émergence de nouvelles symétries dans les CSPs à contraintes de différence ; (2)- améliorer l'efficacité de l'algorithme de détection de ces symétries suite à cet affaiblissement. Nous montrons dans la section 4 comment la propriété de symétrie est exploitée dans un Forward Checking adapté aux CSPs à contraintes de différence. Dans la section 5 nous évaluerons l'efficacité de l'amélioration apportée par des expérimentations réalisées sur des instances du problème de coloriabilité. Dans la section 6 nous discuterons brièvement d'un certain nombre de travaux en rapport avec notre travail. La section 7 conclue cet article.

2 Le Formalisme des CSPs

Un CSP (Constraint Satisfaction Problem en anglais) comme défini dans (Montanari, 1974; Mackworth, 1977) est un quadruplet (X, D, C, R) où :
$X = \{X_1, ..., X_n\}$ est un ensemble de n variables ; $D = \{D_1, ..., D_n\}$ l'ensemble des domaines associés aux variables tel que $\forall X_i \in X$, D_i est l'ensemble des valeurs possibles pour X_i ; $C = \{C_1, ..., C_m\}$ un ensemble de m contraintes reliant deux ou plusieurs variables ; une contrainte est binaire si elle implique deux variables ; $R = \{R_1, ..., R_m\}$ un ensemble de relations correspondant aux contraintes de C, R_i représente la liste des tuples de valeurs permis par la contrainte C_i. Un CSP peut être représenté par un graphe de contraintes $G(X, E)$ dans lequel l'ensemble des sommets X est l'ensemble des variables et chaque arête de E connecte deux variables liées par une même contrainte $C_i \in C$.

Une contrainte binaire est dite de différence si elle force les deux variables X_i et X_j à prendre des valeurs différentes (notation $X_i \neq X_j$). Un CSP à contrainte de différence

(noté NECSP pour Not-Equals CSP en anglais) est un CSP où toutes les contraintes sont des contraintes de différence.

Une instanciation $I = (a_1, ..., a_n)$ est l'affectation de chaque variable X_i à une valeur a_i de son domaine D_i. Une contrainte $C_i \in C$ est satisfaite par I si la projection de I sur les variables impliquées dans C_i est un tuple de R_i. L'instanciation I est consistante si elle satisfait toute contrainte de C, on dit que I est une solution du CSP. Toute instanciation d'un sous ensemble des variables du CSP est dite instanciation partielle. Une instanciation est totale si elle est définie pour toutes les variables du CSP.

Exemple 1
Soit le CSP à contraintes de différence dont le graphe de contraintes est montré dans la figure 1. Les variables sont les sommets $X_1, ..., X_5$ et les domaines sont inclus dans des patates. Chaque arête du graphe reliant deux sommets du graphe, représente la contrainte de différence entre les deux variables correspondantes aux deux sommets.

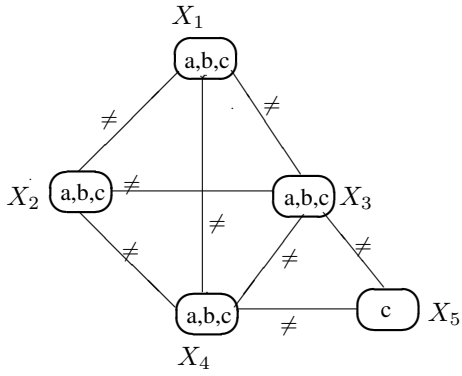


FIG. 1 – Un CSP à contraintes de différence

3 Symétrie dans les CSPs

Nous rappelons dans cette section quelques définitions et propriétés introduites dans (Benhamou, 1994) que nous utilisons pour montrer nos résultats.

Définition 1
Une permutation σ des valeurs des domaines d'un CSP binaire $\mathcal{P} = (V, D, C, R)$ est définit comme :
$\sigma : \cup_{i\in[1,n]} D_i \longrightarrow \cup_{i\in[1,n]} D_i$, tel que $\forall i \in [1, n]$ et $\forall d_i \in D_i, \sigma(d_i) \in D_i$.

Définition 2
Une permutation des valeurs des domaines σ est une symétrie d'un CSP binaire $\mathcal{P} = (V, D, C, R)$ si et seulement si $[\forall R_{ij} \in R, < d_i, d_j > \in tuples(R_{ij}) \Rightarrow < \sigma(d_i), \sigma(d_j) > \in tuples(R_{ij})]$

En d'autres termes, une symétrie du CSP $\mathcal{P}$ est une permutation laissant $\mathcal{P}$ invariant, i.e. $\sigma_R(R_{ij}) = R_{ij}$ [1].

Définition 3
Deux valeurs b_i et c_i d'un domaine d'une variable CSP $v_i \in V$ sont symétriques (notation $b_i \sim c_i$) s'il existe une symétrie σ du CSP $\mathcal{P}$ tel que : $\sigma(b_i) = c_i$ ou $\sigma(c_i) = b_i$

Maintenant nous rappelons comment la symétrie est impliquée dans la consistance des CSPs. Soit I une instanciation des variables d'un CSP $\mathcal{P}$, σ une symétrie de $\mathcal{P}$ et I/σ l'instanciation obtenue en substituant dans I chaque valeur d_i du domaine de chaque variable CSP v_i par $\sigma(d_i)$, formellement : $I/\sigma[v_i] = \sigma(I[v_i]), \forall i \in [1, n]$.

On donne dans ce qui suit une propriété qui peut être utilisée pour calculer une nouvelle solution à partir d'une solution connues en utilisant la symétrie.

Proposition 1
I est une solution de $\mathcal{P}$ si et seulement si I/σ est une solution $\mathcal{P}$.

***Preuve* 1**
Voir (Benhamou, 1994).

On donne maintenant la propriété principale reliant les symétries et la consistance de CSPs :

Théorème 1
Soit b_i et c_i deux valeurs du domaine d'une variable CSP $v_i \in V$. Si b_i et c_i sont symétriques alors b_i participe dans une solution du CSP si et seulement si la valeur c_i participe à une solution du CSP.

***Preuve* 2**
Voir (Benhamou, 1994).

Corollaire 1
Si $d_i \sim \acute{d_i}$ et d_i ne participe pas à aucune solution de $\mathcal{P}$, alors $\acute{d_i}$ ne participe à aucune solution de $\mathcal{P}$.

Le corollaire 1 nous permet de supprimer les valeurs symétriques à une valeur $d_i \in D_i$ sans affecter la consistance du CSP si on a déjà montré que d_i ne participe à aucune solution du CSP.

3.1 La Symétrie dans les NECSPs

Toutes les notions définies précédemment sont valables sur les NECSPs. Dans (Benhamou, 1994) un algorithme est proposé pour la détection des symétries dans les CSPs binaires quelconques. Sa complexité dans le pire des cas est exponentielle. Dans (Benhamou, 2004) il est montré comment les conditions de symétrie peuvent être simplifiées

[1] σ_R : est la généralisation de σ aux relations de R

dans les NECSPs et comment les valeurs symétriques peuvent être calculées efficacement avec un algorithme plus simple. La complexité du nouvel algorithme est linéaire dans la taille du problème. Ce résultat est due à la propriété suivante :

Théorème 2
Soient a_i et b_i deux valeurs d'un domaine D_i d'un NECSP $\mathcal{P}$. Si a_i et b_i apparaissent dans les mêmes domaines des variables non instanciées, alors elles sont symétriques.

***Preuve* 3**
Voir (Benhamou, 2004).

Cette propriété très simple est très utile pour le calcul des valeurs symétriques à une valeurs d'un même domaine. Grâce à cette propriété, on peut déduire que les valeurs a et b du domaine de X_1 du problème illustré dans la figure 1 sont symétriques. En effet elles apparaissent en même temps dans les domaines de X_2, X_3, X_4 et n'apparaissent pas dans le domaine de X_5. Par un raisonnement similaire en peut déduire aussi que les valeurs a et b des domaines des variables X_2, X_3 et X_4 sont symétriques.

3.2 Affaiblissaient de la condition de symétrie

Avant de présenter la nouvelle propriété de symétrie, il nous faut définir ce qu'est un arbre des affectations et ce qu'est un arbre d'échec, d'un processus d'énumération lors de la preuve de consistance d'un CSP.

Définition 4
On appelle arbre des affectations d'un CSP $\mathcal{P}$, un arbre qui regroupe l'historique de toutes les affectations possibles des variables durant la preuve de consistance du CSP $\mathcal{P}$, où les noeuds de l'arbre représentent les variables du CSP et les arêtes partant d'un noeud X_i sont étiquetées par les différentes valeurs utilisées pour instancier la variable X_i.

La forme de l'arbre des affectations dépend de l'ordre dans lequel les variables sont instanciées. Dans l'exemple suivant, on suppose que l'instanciation des variables se fait dans l'ordre $\{X_1, X_2, X_3, X_4, X_5\}$. On supposera aussi que les affectations se font selon l'algorithme d'énumération Forward Checking (Haralik & Elliot, 1980), qui supprime dans les domaines des variables voisines non encore instanciées, les valeurs incompatibles avec l'instanciation de la variable courante.

Exemple 2 (arbre des affectations)
La figure 2 donne l'arbre des affectations du problème de l'exemple 1.

Dans un arbre des affectations, un chemin reliant la racine de l'arbre (qui est la première variable instanciées) à un noeud est en général une instanciation partielle du CSP correspondant. Les variables impliquées dans l'instanciation partielle sont les variables des noeuds par lesquels passe le chemin. Le dernier noeud du chemin correspond à la dernière variable affectée.

A toute instanciation partielle inconsistante, correspondant à un chemin dans l'arbre des affectations, est associé un arbre d'échec que nous définissons comme suite :

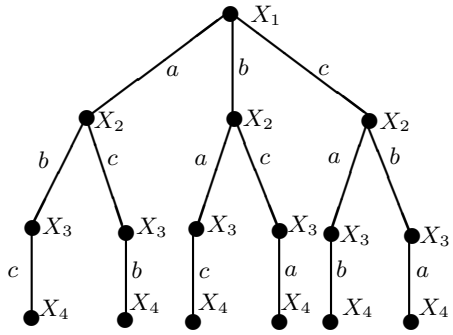


FIG. 2 – Arbre des affectations

Définition 5
Soient T un arbre des affectations définit lors de la preuve de consistance d'un CSP $\mathcal{P}$, $I = (a_1, a_2, ..., a_i)$ une instanciation partielle inconsistante des variables $\{X_1, X_2, ..., X_i\}$ correspondant à un chemin dans T. On appelle arbre d'échec de l'instanciation I, le sous-arbre de T noté $T_{I=(a_1,a_2,...,a_i)}$ tel que :

- la racine de l'arbre T et la racine du sous-arbre $T_{I=(a_1,a_2,...,a_i)}$ sont liées par le chemin correspondant à l'instanciation I ;
- toutes les variables correspondant aux feuilles de $T_{I=(a_1,a_2,...,a_i)}$ ont des domaines vides.

Exemple 3 (arbre d'échec)
Considérons l'arbre des affectations de la figure 2 correspondant au CSP de l'exemple 1. Si on prend comme affectation partielle $I = (b, a)$ qui assigne à X_1 la valeur b et à X_2 la valeur a, alors le sous-arbre d'échec $T_{I=(b,a)}$ de l'instanciation I est montré dans la figure ci-dessous (partie entourée de l'arbre).

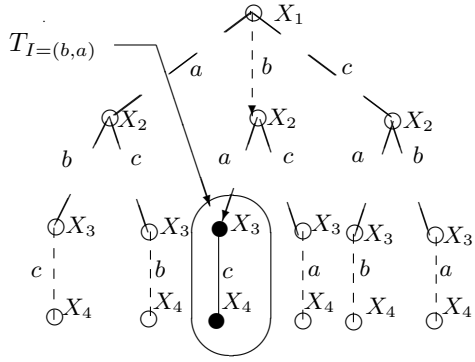


FIG. 3 – Arbre d'échec de l'instanciation partielle $I = (b, a)$

Nous sommes maintenant en mesure d'énoncer la propriété d'affaiblissement de la condition de symétrie, qui représente la contribution principale de ce travail.

Théorème 3
Soient un CSP $\mathcal{P}(X, C, D, R)$, $a_i \in D_i$ et $b_i \in D_i$ deux valeurs du domaine D_i de la variable X_i en cours d'instanciation, $I_0 = (a_1, ..., a_{i-1})$ une instanciation partielle des $i-1$ variables antérieures à X_i telle que l'extension $I = I_0 \bigcup \{a_i\} = (a_1, ..., a_{i-1}, a_i)$ est inconsistante, $T_{I=(a_1,...,a_{i-1},a_i)}$ étant l'arbre d'échec de I et $Var(T_{I=(a_1,...,a_{i-1},a_i)})$ l'ensemble des variables correspondant aux noeuds de $T_{I=(a_1,...,a_{i-1},a_i)}$ On a alors :
Si a_i et b_i apparaissent dans les mêmes domaines des variables de $Var(T_{I=(a_1,...,a_{i-1},a_i)})$ alors l'extension $J = I_0 \bigcup \{b_i\} = (a_1, ..., a_{i-1}, b_i)$ est inconsistante.

***Preuve* 4**
Soit le sous-CSP $\mathcal{P}'(V', C', D', R')$ du CSP $\mathcal{P}(V, C, D, R)$ tel que : $V' = Var(T_{I=(a_1,...,a_{i-1},a_i)}) \cup X_i$ et $C' \subseteq C$ et $D' \subseteq D$ [2] et $R' \subseteq R$ est la restriction de C, D, R aux variables de V'. Par hypothèse, $T_{I=(a_1,...,a_{i-1},a_i)}$ est un arbre d'échec de $\mathcal{P}$. Ce qui implique que l'affectation de X_i à la valeur a_i mène à un échec dans $\mathcal{P}'$. En d'autres mots, a_i ne participe à aucune solution dans $\mathcal{P}'$. Par hypothèse, les deux valeurs a_i et b_i apparaissent dans les mêmes domaines des variables de $Var(T_{I=(a_1,...,a_{i-1},a_i)})$. Ceci revient à dire que a_i et b_i apparaissent dans les mêmes domaines des variables non instanciées du CSP $\mathcal{P}'$. Par application du théorème 2 on déduit que la valeur b_i est symétrique à a_i dans $\mathcal{P}'$. Par application du corollaire 1, on déduit que la valeur b_i ne participe à aucune solution dans $\mathcal{P}'$. Ceci implique que l'instanciation partielle $J = I_0 \bigcup \{b_i\} = (a_1, ..., a_{i-1}, b_i)$ est inconsistante dans $\mathcal{P}$. (CQFD)

Cette nouvelle propriété de symétrie est un affaiblissement de la condition du théorème 2 dans le cas d'instanciation menant à une inconsistance. Le cas d'inconsistance est important, car il permet d'élaguer l'arbre de preuve de consistance des CSPs comme le stipule le corollaire 1. Des symétries non capturées par le théorème 2 peuvent émerger suite à cet affaiblissement. Considérons le CSP de la figure 1. Si on prend l'instanciation partielle inconsistante $I = (b, a)$ des variables X_1 et X_2, alors les deux valeurs a et c du domaine de la variable courante X_2 sont symétriques par application du nouveau théorème 3, alors qu'elles ne le sont pas par application du théorème 2. La branche d'affectation de X_2 à c est coupée dans l'arbre de preuve de consistance grâce au théorème 3. Nous verrons dans la section 4 comment cette coupure est utilisée pour raccourcir l'arbre de preuve de consistance de CSPs.

Remarque 1
Contrairement au théorème 2, le théorème 3 ne peut pas être utilisé pour calculer des solutions symétriques, car ce dernier n'est utilisé que dans le cas d'inconsistance pour éviter des instanciations partielles menant à un échec.

[2] Les domaines de D' sont obtenus par filtrage de domaines de D par l'instanciation I_0.

3.3 Détection des symétries

Nous abordons maintenant le problème de détection des symétries. L'algorithme de la figure 4 calcule les valeurs symétriques à une valeur a_i d'un domaine donné. Ces valeurs forment la classe de symétrie de a_i (notation cl(a_i)).

procédure $Symetrie\_faible(a_i \in D_i, Var(T_{I=(a_1,\ldots,a_i)}))$, **var** cl($a_i$) :classe) ;
Entrée : une valeur $a_i \in D_i$, l'ensemble des variables $Var(T_{I=(a_1,\ldots,a_i)})$
Sortie : la classe $cl(a_i)$ des valeurs symétriques à a_i.
début
cl(a_i) :=$\{a_i\}$;
Pour chaque $d_i \in D_i$-$\{a_i\}$ **faire**
Si pour chaque domaine D_k des variables de $Var(T_{I=(a_1,\ldots,a_i)})$
nous avons
($a_i \in D_k$ et $d_i \in D_k$)
ou ($a_i \notin D_k$ et $d_i \notin D_k$)
alors cl(a_i) :=cl(a_i)$\cup\{d_i\}$;
fin

FIG. 4 – L'algorithme de recherche de symétries dans les NECSPs

Complexité :
Soit n le nombre de variables du NECSP, et d la taille du plus grand domaine. Il est facile de voir que l'algorithme de la figure 4 peut exécuter au plus d fois la première boucle et au plus n fois la deuxième. Donc le calcul de la classe $cl(a_i)$ des valeurs symétriques à a_i est en $\mathcal{O}(nd)$ dans le pire des cas. Cette algorithme a une complexité linéaire dans la taille du NECSP qui est en $\mathcal{O}(n^2 + n * d)$.

En théorie, cet algorithme à la même complexité que celui décrit dans (Benhamou, 2004) dans le pire des cas. Mais dans la pratique, ce nouvel algorithme détecte de nouvelles symétries non détectées par l'ancien algorithme et son efficacité est garantie d'être meilleur car le test d'appartenance des valeurs est effectuée sur un sous ensemble de domaines des variables du CSP. Ces variables sont ceux de l'arbre d'échec $T_{I=(a_1,\ldots,a_i)}$ de l'instanciation partielle considérée.

4 Exploitation de la symétrie dans la résolution des NECSPs

Nous allons montrer maintenant comment exploiter la propriété de symétrie afin de résoudre efficacement des CSPs à contraintes de différence.
Pour des soucis d'efficacité et pour pouvoir comparer objectivement notre version à la version précédente, nous avons choisit d'exploiter l'algorithme de recherche de symétries de la figure 4 dans un *Forward Checking simplifié* (SFC pour simplified Forward Checking en anglais).

Le principe du Forward Checking (Haralik & Elliot, 1980) est basée sur le filtrage des domaines des variables non instanciées en considérant l'instanciation de celles qui précèdent. Quand la variable courante v_i est instanciée à une valeur d_i, le filtrage consiste à supprimer des domaines des variables futures voisines (les variables non-instanciées ayant une contrainte avec la variable courante v_i) toutes les valeurs incompatibles avec la valeur d_i. Si lors du filtrage, le domaine d'une variable future voisine

v_j devient vide, le Forward Checking stoppe le filtrage, ses effets sont annulés, et la variable courante v_i est instanciée avec une nouvelle valeur de son domaine. Si toutes les instanciations possibles pour la variable courante v_i n'aboutissent pas (le filtrage correspondant vide l'un des domaines des variables futures voisines) alors un retour arrière est provoqué et la variable v_{i-1} devient la variable courante. Sinon, si v_i a pu être instanciée sans problèmes alors une nouvelle variable à instancier est choisie parmi les variables futures. Le même processus est alors répété.

Dans le cas des NECSPs, le filtrage est simplifié. Il consiste à supprimer la valeur d_i des domaines des variables futures voisines. Le *Forward Checking simplifié* est donc l'adaptation du Forward Checking aux cas des NECSPs.

La méthode Forward Checking peut avoir des performances très différentes suivant l'ordre d'instanciation des variables. Il existe divers critères pour ordonner les variables (taille des domaines, nombre de contraintes où apparaît une variable). Ce qui a donnée naissance à plusieurs heuristiques, l'une des plus connus est celle qui minimise le rapport

$$r = \frac{\mid D_i \mid}{Degre(v_i)}$$

pour le choix de la prochaine variable à instancier. Cette heuristique connue sous le nom de (DomDeg) est l'une des plus efficaces. Nous l'utiliserons pour le choix de la future variable à instancier.

Le théorème 3 nous permet d'élaguer k-1 branches dans l'arbre de recherche s'il y a k valeurs symétriques et que l'une d'entres elles ne participe à aucune solution. Si $ClasseSym(d_i)$ dénote la classe des valeurs du domaine D_i symétriques à d_i, alors nous considérerions uniquement la valeur d_i, puisque les autres valeurs de $ClasseSym(d_i)$ sont symétriques à d_i.

4.1 Valeurs trivialement symétriques

Certaines valeurs trivialement symétriques peuvent être exploitées sans détection. En effet, toutes les valeurs qui forment l'intersection de tous les domaines d'un NECSP sont trivialement symétriques.

Proposition 2
Si $\mathcal{P}$ est un NECSP ayant n domaines, alors toutes les valeurs dans $\Omega = \cap_{i=1}^{n} D_i$ sont symétriques.

***Preuve* 5**
Le résultat est immédiat, puisque toutes les valeurs dans Ω apparaissent dans tous les domaines, et donc satisfont la condition du théorème 2.

La notion de symétrie triviale est très importante, puisque durant la recherche de solutions (i.e instanciations) les valeurs non utilisées du sous ensemble Ω restent symétriques à chaque noeud de l'arbre de recherche. Nous utilisons uniquement une d'entres elles et les autres sont en fait supprimées par symétrie triviale. Ces symétries sont très utiles dans la résolution des problèmes de coloriage de graphes où toutes les variables ont des

```
Procédure SFC-sym-faible(D, I, i,var L_I) ;
Entrée : l'ensemble des domaines D, I = (d_1, ..., d_i) une instanciation partielle
des variables {v_1, ..., v_i} ; i l'index de la variable courante et L_I la liste des
variables rencontrées lors de la résolution (au début L_I est vide).
var vide :booléen ;
    L_temp : liste ;
début
    si i = n alors [d_1, d_2, ..., d_i] est une solution, STOP ;
    sinon
    début
        vide :=faux ;
        pour chaque v_k ∈ V, tel que C_ki ∈ C, v_k ∈future(v_i) faire
            si non(vide) et d_i ∈ D_k alors
                début
                    D_k :=D_k-{d_i} ;
                    si D_k=∅ alors
                    début
                        annuler les effets du filtrage ;
                        ajouter(v_k, L_I) ;
                        vide :=true ;
                    fin
                fin
        si non(vide) alors
        début
            ajouter(v_i, L_I) ;
            L_temp :=L_I ;
            v_{i+1} :=next-variable(v_i) ;
            répéter
                prendre d_{i+1} ∈ D_{i+1} ;
                D_{i+1} :=D_{i+1} − d_{i+1} ;
                I := [d_1, d_2, ..., d_i, d_{i+1}] ;
                SFC-sym-faible(D, I, i + 1, L_I) ;
                symétrie_faible(d_{i+1} ∈ D_{i+1}, L_I, ClasseSym(d_{i+1})) ;
                D_{i+1} :=D_{i+1} − ClasseSym(d_{i+1}) ;
                L_I :=L_temp ;
            jusqu'à (D_{i+1} = ∅)
        fin
    fin
fin
```

FIG. 5 – La méthode SFC combinée avec l'algorithme de détection de symétries

domaines identiques. L'exploitation gratuite de ces symétries triviales correspondant aux valeurs non utilisées lors d'une instanciation partielle, réduit le coût de détection des symétries, car l'algorithme de détection n'est appliquée qu'aux valeurs utilisées.

La figure 5 représente la procédure SFC qui exploite l'algorithme de symétrie de la figure 4 (notation SFC-sym-faible). $future(v_i)$ représente l'ensemble des variables non instanciées qui restent après l'instanciation de v_i et $next\text{-}variable()$ est la fonction qui permet de choisir la prochaine variable à instancier suivant l'heuristique (DomDeg).

5 Expérimentations

Nous allons maintenant évaluer les performances de notre implémentation. Les tests ont été effectués sur les problèmes de coloration de graphes : des instances issus du 2^{eme} challenge de Dimacs (`http ://dimacs.rutgers.edu/Challenges/`), ainsi que des problèmes générés de façon aléatoire. Le problème de coloration de graphes

consiste à colorer les sommets d'un graphe sans que deux sommets reliés par une arête ne soit colorés avec une même couleur. Ce problème s'exprime de façon triviale en CSP à contraintes de différence. Nous allons tester et comparer le Forward Checking simplifié avec détection des valeurs symétriques (SFC-sym) comme définit dans (Benhamou, 2004) et le Forward Checking simplifié avec détection des symétries (SFC-sym-faible). Les indicateurs de complexité sont le nombre de noeuds et le temps d'exécution. Les tests ont été réalisés sur une machine équipé d'un processeur AMD Athlon XP 2200+ avec 512 MB.

5.1 Problèmes aléatoires de coloration de graphes

Les problèmes aléatoires de coloration de graphes sont générés en fixant les paramètres suivant : (1) n le nombre de sommets (les variables), (2) Cls le nombre de couleurs (les domaines) et (3) d la densité qui est un nombre entre 0 et 1 exprimant le rapport :

$$d = \frac{nombre\ de\ contraintes(nombre\ d'aretes\ dans\ le\ graphe\ des\ contraintes)}{le\ nombre\ totale\ de\ contraintes\ possibles}$$

Pour n, Cls et d fixés, le nombre de noeuds ainsi que le temps d'exécution sont calculés sur la moyenne des résultats obtenus à partir de 100 instances générées aléatoirement.

Pb	SFC-sym		SFC-sym-faible		
Cls	Noeuds	Temps	Noeuds	Temps	%Cons
13	69	0.0	67	0.0	0.0
14	479	0.0	467	0.0	0.0
15	7014	0.0	6843	0.0	0.0
16	181448	2.8	177024	2.7	17.0
17	1894312	32.5	1837245	32.0	45.0
18	288708	4.6	279559	4.5	100.0
19	7760	0.1	7389	0.1	100.0
20	252	0.0	239	0.0	100.0
21	102	0.0	101	0.0	100.0

TAB. 1 – instances de coloration de graphes avec $n = 100$ et $d = 0.5$

Pb	SFC-sym		SFC-sym-faible		
Cls	Noeuds	Temps	Noeuds	Temps	%Cons
35	114061	2.6	110578	2.6	0.0
36	1645219	39.7	1585537	38.4	0.0
37	5327514	110.3	5152504	106.8	15.0
38	18553369	374.8	17992800	365.5	55.0
39	2638689	52.7	2520893	50.25	95.0
40	437893	7.15	423600	6.9	100.0
41	163319	2.4	157073	2.35	100.0
42	14743	0.2	14008	0.1	100.0

TAB. 2 – instances de coloration de graphes avec $n = 100$ et $d = 0.9$

Les Tables 1 et 2 donnent les résultats des deux versions SFC-sym et SFC-sym-faible pour les problèmes de coloration de graphes générés aléatoirement avec $n = 100$ et les densités $d = 0.5$ et $d = 0.9$. Les tables donnent pour chaque méthode le nombre

de couleurs du problème (Cls), le nombre de noeuds moyens (noeuds), le temps moyen d'exécution en seconde (temps) et le pourcentage de consistance (%Cons). Nous pouvons voir que la méthode SFC-sym-faible génére en moyenne moins de noeuds que la méthode SFC-sym et qu'elle permet d'améliorer de quelques pourcents le temps moyen au voisinage du pic de difficulté. Le gain n'est pas très significatif sur les problèmes aléatoires, mais en aucun cas la méthode SFC-sym-faible est plus lente que SFC-sym.

5.2 Problèmes de coloration de graphes de Dimacs

Pb		SFC		SFC-sym		SFC-sym-faible	
instance	k	Noeuds	Temps	Noeuds	Temps	Noeuds	Temps
queen8_8	9	-	-	7154238	42.8	7085774	42.8
le450_5a	5	168052	5.5	125569	4.7	124462	4.8
le450_5b	5	9908	0.3	9904	0.4	9875	0.4
myciel5	6	770786	2.3	32179	0.1	31550	0.1
mulsol.i.1	49	-	-	-	-	70358	1.8
mulsol.i.4	31	-	-	3638	0.0	3638	0.0
fpsol2.i.1	65	-	-	4354300	150.0	9684	1.0
fpsol2.i.3	30	-	-	-	-	385036	13.5
zeroin.i.1	49	-	-	362	0.0	263	0.0
zeroin.i.3	30	-	-	254	0.0	236	0.0
school1	14	-	-	144421	7.4	61076	3.4
school1_nsh	14	-	-	797	0.0	645	0.0
miles750	31	-	-	51017	0.4	294	0.0
miles1500	73	-	-	6030193	104.8	14206	0.3
1-Fullins_4	5	91402	0.4	21968	0.1	17759	0.1
2-Fullins_3	5	294687	0.5	29150	0.1	5857	0.0

TAB. 3 – Problèmes de coloration de graphes de Dimacs

La Table 3 montre les résultats des deux méthodes sur un certain nombre de problèmes de coloration de graphes issus de benchmarks de Dimacs. Ces problèmes sont connus pour être difficiles à résoudre. Nous cherchons pour chaque problème le nombre minimal k de couleurs nécessaire pour colorer les sommets d'un graphe donné (le nombre chromatique). La recherche du nombre chromatique consiste à prouver la consistance du problème pour k couleurs (existence d'une k-coloration du graphe), ensuite à prouver l'inconsistance du problème pour $k-1$ couleurs. Le symbole "-" signifie que le programme correspondant n'a pas retourné de réponse après 12 heures de calculs.

Afin de montrer l'importance des symétries dans la résolution, nous avons aussi fait apparaître les résultats obtenus sur les mêmes problèmes en utilisant l'algorithme SFC de base. C'est à dire sans élimination de symétries (Colonne SFC). Mise à part quelques problèmes triviaux, on voit bien que la méthode SFC est incapable de résoudre les problèmes de Dimacs.

On remarque que seule la méthode SFC-sym-faible est capable de résoudre les problèmes ("mulsol.i.1", "fpsol2.i.3"). Il aussi intéressant de voir que la méthode SFC-sym-faible améliore sensiblement les performances de la méthode SFC-sym sur les problèmes ("miles1500", "fpsol2.i.1").

Par ailleurs tous les problèmes résolus efficacement avec la méthode SFC-sym, sont également résolus de manière aussi efficace avec la méthode SFC-sym-faible.

6 Comparaison avec d'autres travaux

Dans (Hentenryck *et al.*, 2003) les auteurs ont étudié trois classes de CSPs où les symétries sont traitables. La classe des CSPs à valeurs Interchangeables (ICSPs) est en relation avec notre travail. En effet, quand tous les domaines d'un NECSP sont identiques (comme c'est le cas dans le problème de coloration de graphes), il devient un ICSP. Pour ce cas particulier, la technique d'élimination des valeurs symétriques utilisée dans (Hentenryck *et al.*, 2003) pour la classe ICSP semble être équivalente à l'élimination des symétries globales décrite précédemment. Mais, en général les NECSPs ne sont pas des ICSPs et les symétries détectées par la procédure de la figure 4 ne sont pas capturées par la technique d'élimination des symétries dans les ICSPs.

Récemment, dans (Ramani *et al.*, 2004) les auteurs proposent une approche statique pour l'élimination des symétries dans les problèmes de coloration de graphes. Les problèmes sont réduit en un ensemble de contraintes hybrides formées de contraintes booléennes et de contraintes pseudo booléennes obtenues par une réduction du problème initial en 0-1 ILP (Integer Linear Programming). Deux sortes de symétries sont exploitées dans cette approche : les symétries dépendantes et les symétries indépendantes des instances. Cette approche est statique alors que la notre est dynamique. Notre méthode semble être plus efficace pour la résolution des problèmes de coloration de graphes.

Dans (Puget, 2004), l'auteur étudie l'élimination des symétries de variables liées par une contrainte globale AllDiff. Il nous semble que c'est un cas particulier de NECSPs, puisque la contrainte AllDiff entre les variables du CSP peut être représentée par un NECSP dont le graphe de contrainte est complet. La technique est statique et consiste à rajouter au problème initial un nombre linéaire de contraintes binaires afin d'éliminer toutes les symétries liées aux variables. Notre approche est dynamique et porte sur la symétrie entre les valeurs, il serait donc intéressant de voir ce qui se passerait, dans le cas de NECSPs complets , si on combinait notre approche pour l'élimination des valeurs symétriques avec celle de Puget.

7 Conclusion

Dans ce travail nous avons amélioré les propriétés de symétrie dans les contraintes de différence, nous avons mis au point un algorithme de détection plus efficace et qui permet de calculer plus de symétries. L'algorithme de détection de symétries a été utilisé dans un algorithme de type Forward Checking adapté aux contraintes de différence. L'étude expérimentale a montré que le nouvel algorithme SFC-sym-faible est plus efficace que SFC-sym. Cette étude nous a aussi permis de voir que les symétries sont d'un grand apport pour la résolution des NECSPs.

Ce travail soulève plusieurs points à discuter qui pourront être développé dans le futur. Certains points nous semblent particulièrement intéressants : (1)- généraliser la nouvelle propriété de symétrie au cas général de CSP ; (2)- combiner des algorithmes de recherche de cliques, ou de décomposition de CSPs avec notre méthode pour améliorer la résolution des problèmes de coloriabilité de graphes ; (3)- exploiter d'autres formes de symétries comme la symétrie de variables ; (4)- améliorer l'heuristique de choix

de variables à fin de maximiser le nombre de symétries supprimées.

Références

BENHAMOU B. (1994). Study of symmetry in constraint satisfaction problems. *In the working notes of the workshop PPCP'94*.

BENHAMOU B. (2004). Symmetry in not-equals binary constraint networks. *SymCon'04 : 4th International Workchop on Symmetry and Constraint Satisfaction Problems*.

CRAWFORD J., GINSBERG M. L., LUCK E. & ROY A. (1996). Symmetry-breaking predicates for search problems. In *KR'96 : Principles of Knowledge Representation and Reasoning*, p. 148–159. San Francisco, California : Morgan Kaufmann.

DECHTER R. & PEARL J. (1987). The cycle-cutset method for improving search performance in ai applications. *Proceedings of the Third IEEE Conference on AI Applications*.

DECHTER R. & PEARL J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, **38**, 353– 366.

FAHLE T., SCHAMBERGER S. & SELLMANN M. (2001). Symmetry breaking. In *International conference on constraint programming*, volume 2239 of *LNCS*, p. 93–108 : Springer Verlag.

FOCACCI F. & MILANO M. (2001). Global cut framework for removing symmetries. In *International conference on constraint programming*, volume 2239 of *LNCS*, p. 77–82 : Springer Verlag.

FREUDER E. (1978). Synthesing constraint expressions. *CACM*, **21**(11), 958–966.

FREUDER E. (1991). Eliminating interchangeable values in constraints satisfaction problems. *Proc AAAI-91*, p. 227–233.

HARALIK R. M. & ELLIOT G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence 14*, p. 263–313.

HENTENRYCK P. V., FLENER P., PEARSON J. & ARGEN M. (2003). Tractable symmetry breaking for csps with interchangeable values. In *IJCAI*, p. 277–282.

JÉGOU F. (1990). Cyclic-clustering : A compromise between tree-clustering and cycle-cutset method for improving search efficiency. *ECAI1990*, p. 369–371.

JÉGOU F. (1993). Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. *AAAI1993*, p. 731–736.

MACKWORTH A. (1977). Consistency in networks of relations. *Artificial Intelligence 8*, p. 99–118.

MONTANARI U. (1974). Networks of constraints : Fundamental properties and applications to picture processing. *Information Science 7*, p. 95–132.

PUGET J. (2001). Symmetry breaking revisited. In *International conference on constraint programming*, volume 2470 of *LNCS*, p. 446–461 : Springer Verlag.

PUGET J. (2004). Breaking symmetries in all diffrent problems. *SymCon'04 : 4th International Workchop on Symmetry and Constraint Satisfaction Problems*.

RAMANI A., ALOUL F., MARKOV I. & SAKALLAK K. (2004). Breaking instance-independent symmetries in exact graph coloring. *DATE*, p. 324– 329.

Similarité de graphes : une mesure générique et un algorithme tabou réactif

Sébastien Sorlin et Christine Solnon

LIRIS, CNRS UMR 5205, bât. Nautibus, Université de Lyon I
43 Bd du 11 novembre, 69622 Villeurbanne cedex, France
{sebastien.sorlin,christine.solnon}@liris.cnrs.fr

Résumé :

De nombreuses applications nécessitent de mesurer la similarité d'objets, *e.g.*, la reconnaissance d'images, la recherche d'information, le raisonnement à partir de cas... Lorsque les objets sont représentés par des graphes, ce problème se ramène à mesurer la similarité de graphes. Différents types d'appariements de graphes, définissant chacun une mesure de similarité différente, ont été proposés : l'isomorphisme de (sous-)graphes, les appariements à tolérance d'erreur, la distance d'édition de graphes... Un premier objectif de cet article est de montrer que ces mesures peuvent être vues comme des cas particuliers d'une mesure de similarité introduite dans Champin & Solnon (2003). Initialement définie pour la comparaison d'objets de conception, cette mesure est basée sur un appariement multivoque de deux graphes (*i.e.*, un appariement où chaque sommet peut être apparié à plusieurs sommets) ce qui permet la comparaison d'objets décrits à des niveaux de granularité différents ou d'images sur- ou sous-segmentées. Nous nous intéressons ensuite au calcul de cette similarité et nous décrivons deux algorithmes : un algorithme glouton capable de calculer rapidement une approximation de la similarité de deux graphes et une recherche locale taboue réactive améliorant cette approximation. Quelques résultats expérimentaux sont donnés.

Mots-clés : Similarité, Appariement de graphes, Isomorphisme de graphes, Recherche locale, Méta-heuristique taboue

1 Introduction

Les graphes sont souvent utilisés pour modéliser des objets structurés. Dans Champin & Solnon (2003), des objets de conception sont modélisés par des graphes étiquetés où les sommets représentent les composants des objets et les arcs représentent les relations entre ces composants. Afin de distinguer les différents types de composants et de relations, des étiquettes sont ajoutées aux sommets et aux arcs. Les graphes permettent également la représentation d'images (*e.g.*, Ambauen *et al.* (2003)) : un sommet représente une région de l'image et peut être étiqueté par ses propriétés (couleur, taille....) et un arc représente une relation binaire entre deux régions et peut être étiqueté par la nature de la relation (connectivité, position relative...).

Comparer deux objets revient alors à comparer deux graphes, *i.e.*, mettre en relation leurs sommets (les apparier) afin d'identifier leurs points communs. Cette comparaison peut se faire à travers la recherche d'une relation d'isomorphisme de (sous-)graphes afin de montrer l'existence d'une relation d'équivalence ou d'inclusion entre les deux graphes. Cependant, deux objets "similaires" ne sont pas nécessairement "identiques" et présumer de l'existence d'une relation permettant de retrouver tous les sommets et tous les arcs est généralement utopique. Par conséquent, des techniques de comparaison de graphes à tolérance d'erreurs telles que la recherche du plus grand sous-graphe commun ou la "graph edit distance" ont été proposées (Bunke (1997); Conte *et al.* (2004)).

Plus récemment, trois articles différents proposent de franchir une étape supplémentaire dans la comparaison de deux graphes en introduisant la notion d'appariement multivoque, *i.e.*, un appariement permettant d'associer un sommet d'un graphe à un ensemble de sommets de l'autre graphe :

- Dans Champin & Solnon (2003), des graphes sont utilisés pour représenter des objets de conception. Dans ce contexte et selon le niveau de granularité de représentation des objets, un seul composant d'un objet peut jouer le rôle d'un ensemble de composants d'un autre objet. Pour en tenir compte, les auteurs utilisent des appariements multivoques permettant de relier un sommet d'un graphe à un ensemble de sommets de l'autre graphe.
- Dans Boeres *et al.* (2004), l'appariement de graphes est utilisé pour comparer des images segmentées du cerveau à un modèle du cerveau. Le modèle, d'aspect schématique, est "correctement" segmenté alors que les images, bruitées, sont généralement sur-segmentées. Il n'existe donc pas de bijection entre les régions du modèle et les régions de l'image. Les auteurs utilisent une mesure de similarité entre deux images basée sur des appariements multivoques où un sommet du schéma peut être apparié à plusieurs sommets de l'image.
- Dans Ambauen *et al.* (2003), une nouvelle distance d'édition de graphes introduisant deux nouvelles opérations –la fusion et l'éclatement de sommets– est proposée. Ces deux nouvelles opérations permettent de prendre en compte le fait que les images à comparer sont généralement sur- ou sous- segmentées.

Motivation et plan. Un premier objectif de cet article est de mettre en évidence les points communs entre les appariements proposés dans ces trois articles. Un second objectif est de proposer des algorithmes pour la recherche du meilleur appariement multivoque de deux graphes. La partie 2 introduit brièvement la mesure de similarité proposée par Champin & Solnon (2003). En section 3, on montre que cette mesure générique peut s'instancier en d'autres mesures de similarité. En section 4 nous nous intéressons au problème du calcul de cette similarité : nous proposons un algorithme glouton calculant rapidement une approximation de la similarité ainsi qu'une recherche locale taboue réactive améliorant les résultats obtenus par l'algorithme glouton. Finalement, la section 5 présente quelques résultats expérimentaux.

2 Une mesure de similarité de graphes multi-étiquetés

Un **graphe orienté** est défini par un couple $G = (V, E)$, où V est un ensemble fini de sommets et $E \subseteq V \times V$ est un ensemble fini d'arcs orientés. Dans un graphe multi-

étiqueté, les sommets et les arcs sont associés à des étiquettes décrivant leurs propriétés. Etant donnés L_V (resp. L_E) un ensemble d'étiquettes de sommets (resp. d'arcs), un **graphe multi-étiqueté** est défini par un triplet $G = \langle V, r_V, r_E \rangle$ tel que :

- V est un ensemble de sommets,
- $r_V \subseteq V \times L_V$ est une relation associant les sommets à leurs étiquettes, *i.e.*, r_V est l'ensemble des couples (v, l) tels que le sommet v a pour étiquette l,
- $r_E \subseteq V \times V \times L_E$ est une relation associant les arcs à leurs étiquettes, *i.e.*, r_E est l'ensemble des triplets (v, v', l) tels que l'arc (v, v') a pour étiquette l. Sans perte de généralité, on supposera que chaque arc possède au moins une étiquette. Par conséquent, l'ensemble E des arcs est défini par $E = \{(v, v') | \exists l, (v, v', l) \in r_E\}$.

Les tuples de r_V et r_E constituent les caractéristiques de G. L'ensemble $descr(G) = r_V \cup r_E$ contient toutes ces caractéristiques des sommets et des arcs de G et décrit entièrement le graphe G.

Nous introduisons maintenant la mesure de similarité de graphes de Champin & Solnon (2003) ; nous invitons le lecteur à se référer à cet article pour plus de détails. Cette mesure de similarité est définie pour deux graphes multi-étiquetés $G = \langle V, r_V, r_E \rangle$ et $G' = \langle V', r_{V'}, r_{E'} \rangle$ définis sur les mêmes ensembles L_V et L_E d'étiquettes et tels que $V \cap V' = \emptyset$.

Pour mesurer la similarité de deux graphes, une première étape consiste à apparier leurs sommets. L'appariement considéré ici est multivalent, *i.e.*, chaque sommet d'un graphe est apparié à un ensemble éventuellement vide de sommets de l'autre graphe. Plus formellement, un **appariement multivoque** de deux graphes G et G' est une relation $m \subseteq V \times V'$ contenant tous les couples $(v, v') \in V \times V'$ tels que le sommet v est apparié au sommet v'.

Etant donné un appariement m, l'étape suivante consiste à identifier l'ensemble des caractéristiques communes aux deux graphes par rapport à m. Cet ensemble contient toutes les caractéristiques des sommets (resp. arcs) de G et de G' appariés dans m à au moins un sommet (resp. arc) ayant la même caractéristique. Plus formellement, l'ensemble $descr(G) \sqcap_m descr(G')$ des caractéristiques communes à G et G' par rapport à l'appariement m est défini par :

$$\begin{aligned} descr(G) \sqcap_m descr(G') \quad \doteq \quad & \{(v,l) \in r_V \mid \exists (v,v') \in m,\ (v',l) \in r_{V'}\} \\ \cup \quad & \{(v',l) \in r_{V'} \mid \exists (v,v') \in m,\ (v,l) \in r_V\} \\ \cup \quad & \{(v_i,v_j,l) \in r_E \mid \exists (v_i,v_i') \in m, \exists (v_j,v_j') \in m, (v_i',v_j',l) \in r_{E'}\} \\ \cup \quad & \{(v_i',v_j',l) \in r_{E'} \mid \exists (v_i,v_i') \in m, \exists (v_j,v_j') \in m, (v_i,v_j,l) \in r_E\} \end{aligned}$$

Etant donné un appariement multivoque m, nous devons aussi identifier l'ensemble des sommets éclatés (les *splits*), *i.e.*, l'ensemble des sommets appariés à plus d'un sommet. Chaque sommet éclaté v est associé à l'ensemble s_v des sommets auquel il est apparié :

$$\begin{aligned} splits(m) = \quad & \{(v, s_v) \quad \mid v \in V, \quad s_v = \{v' \in V' | (v,v') \in m\}, \quad |s_v| \geq 2\} \\ \cup \quad & \{(v', s_{v'}) \quad \mid v' \in V', \quad s_{v'} = \{v \in V | (v,v') \in m\}, \quad |s_{v'}| \geq 2\} \end{aligned}$$

La **similarité** de G et G' par rapport à un appariement m est alors définie par :

$$sim_m(G,G') \quad = \quad \frac{f(descr(G) \sqcap_m descr(G')) - g(splits(m))}{f(descr(G) \cup descr(G'))} \qquad (1)$$

où f et g sont deux fonctions dépendantes de l'application considérée. Par exemple, si f est la fonction cardinalité et g la fonction nulle, la similarité est alors proportionnelle au

nombre de caractéristiques communes par rapport au nombre total de caractéristiques. Si g est une fonction de cardinalité, alors la similarité décroîtra proportionnellement au nombre de sommets éclatés.

Finalement, la similarité $sim(G, G')$ de deux graphes G et G' est définie comme la plus grande similarité possible, *i.e.*, celle obtenue par le meilleur appariement :

$$sim(G, G') = \max_{m \subseteq V \times V'} \frac{f(descr(G) \sqcap_m descr(G')) - g(splits(m))}{f(descr(G) \cup descr(G'))} \quad (2)$$

3 Généricité de cette mesure

La mesure de similarité décrite en section 2 a été initialement proposée pour comparer des objets de conception. Cependant, cette mesure est générique dans le sens où elle est paramétée par les fonctions f et g. Dans cette section, nous montrons comment ces fonctions peuvent être définies pour instancier cette mesure générique en d'autres mesures de similarité couramment utilisées. Ces mesures sont souvent définies pour des graphes non-étiquetés. Nous supposerons donc qu'un graphe non-étiqueté est un cas particulier de graphe étiqueté où tous les sommets (resp. arcs) ont la même étiquette l_v (resp. l_e).

Isomorphisme de graphes. Un graphe $G = (V, E)$ est isomorphe à un graphe $G' = (V', E')$ ssi $|V| = |V'|$ et s'il existe une fonction bijective $\phi : V \rightarrow V'$ telle que $(v_1, v_2) \in E$ ssi $(\phi(v_1), \phi(v_2)) \in E'$. Autrement dit, le problème consiste à trouver un appariement univoque permettant de retrouver tous les sommets et tous les arcs des deux graphes.

Si les fonctions f et g de la formule (2) sont définies comme la fonction de cardinalité, alors $sim(G, G')$=1 ssi il existe un appariement m tel que $descr(G) \sqcap_m descr(G') = descr(G) \cup descr(G')$ (*i.e.*, m permet de retrouver toutes les caractéristiques de G et G') et $splits(m) = \emptyset$, (*i.e.*, m est univoque). Autrement dit, $sim(G, G') = 1$ ssi G et G' sont isomorphes.

Isomorphisme de sous-graphes partiel. Un graphe $G = (V, E)$ est un sous-graphe partiel du graphe $G' = (V', E')$ ssi $|V| \leq |V'|$ et s'il existe une fonction injective $\phi : V \rightarrow V'$ telle que $(v_1, v_2) \in E \Rightarrow (\phi(v_1), \phi(v_2)) \in E'$. Autrement dit, le problème consiste à trouver un appariement univoque des sommets de G aux sommets de G' permettant de retrouver toutes les caractéristiques de G.

Si la fonction g de la formule (2) est la fonction cardinalité et si f est une fonction ne comptant que les caractéristiques de G, *i.e.*, f est définie comme une somme pondérée où le poids des caractéristiques de G (resp G') est 1 (resp 0), alors $sim(G, G') = 1$ ssi il existe un appariement m tel que $descr(G) \subseteq descr(G) \sqcap_m descr(G')$ (*i.e.*, m permet de retrouver toutes les caractéristiques de G) et $splits(m) = \emptyset$ (*i.e.*, m est univoque). Autrement dit, $sim(G, G') = 1$ ssi il existe un isomorphisme de sous-graphes partiel de G dans G'.

Isomorphisme de sous-graphes. Le problème de l'isomorphisme de sous-graphes est un cas particulier du problème de l'isomorphisme de sous-graphes partiel où la condition suivante est ajoutée : tous les couples de sommets de G qui ne sont pas reliés par un arc doivent être appariés à des sommets de G' qui ne sont également pas reliés par un arc (*i.e.*, $\forall (v_1, v_2) \in V^2, (v_1, v_2) \notin E \Rightarrow (\phi(v_1), \phi(v_2)) \notin E'$).

Pour vérifier cette condition, il faut ajouter à tous les couples de sommets non reliés par un arc une étiquette "pas-un-arc" puis s'assurer que le meilleur appariement retrouve ces étiquettes. Plus formellement, étant donné un graphe $G = (V, E)$, nous définissons le graphe étiqueté $G_{label} = (V, r_V, r_E)$ tel que $r_V = \{(v, l_v)|v \in V\}$ et $r_E = \{(u, v, l_e)|(u, v) \in E\} \cup \{(u, v, l_{notE})|(u, v) \in V \times V - E\}$. Si les fonctions f et g sont définies comme pour le problème de l'isomorphisme de sous-graphes partiel, $sim(G_{label}, G'_{label}) = 1$ ssi il existe un isomorphisme de sous-graphes de G dans G'.

Plus grand sous-graphe partiel commun ($mcps$). Le $mcps$ de deux graphes G et G' est le plus grand graphe (par rapport au nombre de sommets et d'arcs) qui soit isomorphe à des sous-graphes partiels de G et de G'.

Définissons la fonction f de la formule (2) comme une fonction de cardinalité et la fonction g telle que tout éclatement de sommet soit interdit (*i.e.*, $g(S) = +\infty$ si $S \neq \emptyset$ et $g(\emptyset) = 0$). L'appariement m qui maximise la formule (1) est alors l'appariement permettant de retrouver un maximum de caractéristiques de sommets et d'arcs tout en interdisant les éclatements de sommets. Cet appariement m correspond donc à un $mcps$.

Plus grand sous-graphe commun (mcs). En s'inspirant du problème de l'isomorphisme de sous-graphes non partiel, il est possible de résoudre le problème de la recherche du plus grand sous-graphe non partiel commun (mcs) à deux graphes. Pour cela, il faut définir la fonction g de telle façon que les éclatements de sommets soient interdits (*i.e.*, $g(S) = +\infty$ si $S \neq \emptyset$ et $g(\emptyset) = 0$). La fonction f doit "vérifier" que seuls des couples de sommets de même types (*i.e.*, reliés ou pas à un arc) sont appariés entre eux. Autrement dit, la fonction f doit retourner une valeur nulle s'il existe un couple de sommets (u, v) de G (resp. de G') apparié à un couple de sommet (u', v') de G' (resp. de G) tel que la relation entre u et v (relié ou pas par un arc) n'est pas la même que la relation entre u' et v'. Dans tous les autres cas, f doit retourner une valeur proportionnelle au nombre de sommets appariés. De façon plus formelle, en utilisant les graphes G_{label} et G'_{label} définis pour l'isomorphisme de sous-graphes non partiel, $f(S) = 0$ si $\exists(u, v) \in V_1^2 \cup V_2^2$ tel que $\{(u, l_v), (v, l_v)\} \subseteq S$ (*i.e.*, les sommets u et v sont appariés) et que $\{(u, v, l_e), (u, v, l_{notE})\} \cap S = \emptyset$ (*i.e.*, aucune propriété reliant u à v n'a été retrouvée). Dans tous les autres cas, $f(S) = |\{u|(u, l_v) \in S\}|$. Avec de telles fonctions f et g, le meilleur appariement m correspond au mcs.

Distance d'édition de graphes (ged). La distance d'édition (ged) entre deux graphes G et G' est le coût minimal pour transformer G en G'. Pour cette transformation, on dipose de six opérations élémentaires : l'insertion, la suppression et le réétiquetage de sommets et d'arcs. Chaque opération a un coût. L'ensemble d'opérations le moins couteux permettant de transformer G en un graphe isomorphe à G' définit la distance d'édition entre G et G'. Bunke (1997) montre que la distance d'édition est un concept très proche de la mesure de similarité basée sur le plus grand sous-graphe partiel commun à deux graphes : elle est donc également très proche de la mesure de similarité de la formule (2).

Si les graphes ne sont pas étiquetés, seules les opérations de suppression et d'insertion sont utilisées. Nous pouvons remarquer que, étant donné un appariement m, l'ensemble des caractéristiques de sommets et d'arcs contenues dans $descr(G) - (descr(G) \sqcap_m descr(G'))$ (resp. dans $descr(G') - (descr(G) \sqcap_m descr(G'))$) correspond à l'ensemble des opérations de suppression (resp. d'insertion) d'arcs et de sommets néces-

saires pour rendre le graphe G isomorphe au graphe G'. Choisissons g de telle façon que les éclatements de sommets soient interdits et la fonction f de la formule (2) comme une fonction de pondération où les poids sont définis en fonction des coûts des transformations élémentaires. L'appariement m qui maximise la formule (1) définit l'ensemble des opérations élémentaires qui minimise la distance d'édition.

Si nous considérons maintenant la distance d'édition appliquée à des graphes mono-étiquetés (*i.e.,* des graphes où chaque sommet et chaque arc possède une seule étiquette), les opérations de substitution (moins coûteuses qu'une suppression suivie d'une insertion) peuvent s'appliquer. Cependant, lorsque les sommets (resp. les arcs) sont mono-étiquetés, les informations contenues dans l'ensemble $descr(G) \sqcap_m descr(G')$ des caractéristiques communes ne permettent pas de différencier le cas où deux sommets (resp. arcs) n'ayant pas la même étiquette sont appariés du cas où ces deux sommets (resp. arcs) ne sont pas appariés (dans les deux cas, aucune des deux étiquettes n'est retrouvée). Afin de détecter les substitutions, il est nécessaire d'ajouter à tous les sommets (resp. tous les arcs) une étiquette commune l_v (resp. l_e) en plus de leur étiquette d'origine. Ces étiquettes permettent alors de savoir si un sommet (resp. un arc) est apparié ou non. Si l'étiquette l_v d'un sommet u (resp. l'étiquette l_e d'un arc (u, v)) n'appartient pas à $descr(G) \sqcap_m descr(G')$, alors le sommet u (resp. l'arc (u, v)) a été supprimé ou inséré. A contrario, lorsque cette étiquette l_v (resp. l_e) est retrouvée, soit il y a eu substitution – et l'étiquette d'origine de u (resp. (u, v)) n'est pas retrouvée – soit il n'y a pas eu d'opération de transformation sur u (resp. (u, v)). La fonction de pondération f peut être choisie de telle sorte que les poids reproduisent fidèlement les coûts de tous les types de transformation élémentaire.

Distance d'edition étendue. Afin de comparer des images sur- et sous- segmentées, Ambauen *et al.* (2003) propose une distance d'édition de graphe étendue autorisant deux nouvelles opérations par rapport à la distance d'édition classique : l'éclatement de sommets –reliant un sommet de G à plusieurs sommets de G'– et la fusion de sommets –reliant plusieurs sommets de G à un même sommet de G'.

Etant donné un appariement m, l'ensemble des couples $(v, s_v) \in splits(m)$ tels que $v \in G$ correspond aux opérations d'éclatement de sommets et l'ensemble des couples $(v', s_{v'})$ tels que $v' \in G'$ correspond aux fusions de sommets. Dés lors, si la fonction g de la formule (2) est définie comme une fonction de pondération où les poids correspondent aux coûts d'éclatement et de fusion de sommets et si la fonction f et les graphes sont définis comme pour la distance d'édition non étendue, alors l'appariement qui maximise la formule (1) permet de calculer la distance d'édition étendue telle que définie dans Ambauen *et al.* (2003).

Appariement non bijectif. Ce problème est introduit dans Boeres *et al.* (2004) afin de trouver le meilleur appariement entre les régions d'une image schématique d'un cerveau et les régions d'une image réelle du cerveau. L'image réelle est bruitée (contrairement à l'image schématique) et donc sur-segmentée si bien qu'une région du schéma correspond souvent à plusieurs régions de l'image. Etant donné un graphe modèle $G = (V, E)$ et un graphe image $G' = (V', E')$, les appariements autorisés sont définis par une fonction $\phi : V \rightarrow \wp(V')$ associant à chaque sommet de V un ensemble non vide de sommets de V' et telle que (i) chaque sommet de V' est associé à exactement un sommet de V, (ii) quelques couples de sommets (v, v') jugés trop différents sont interdits (*i.e.,* $v' \notin \phi(v)$)

et (iii), le sous-graphe de G' induit par chaque ensemble $\phi(v)$ doit être connexe (afin de ne fusionner que des régions adjacentes). Un poids (éventuellement négatif) $s^v(v_i, v'_i)$ (resp. $s^e(e_i, e'_i)$) est associé à chaque couple de sommets $(v_i, v'_i) \in V \times V'$ (resp. à chaque couple d'arcs $(e_i, e'_i) \in E \times E'$). L'objectif est de trouver un appariement respectant les contraintes et maximisant la somme des poids des couples de sommets et d'arcs appariés.

Les fonctions f et g peuvent être définies de telle façon que l'appariement m qui maximise la fonction (2) corresponde au meilleur appariement au sens de Boeres *et al.* (2004). Afin de prendre en compte le fait que des poids sont associés aux couples de sommets et d'arcs et que la condition (ii) est respectée, il convient d'ajouter des étiquettes aux sommets (resp. aux arcs) de telle façon qu'une étiquette $l_{(v,v')}$ (resp. $l_{(e,e')}$) appartienne à l'ensemble $descr(G) \sqcap_m descr(G')$ des caractéristiques communes ssi v est apparié à v' (resp. e à e'). Pour cela, il suffit d'ajouter à tous les sommets $v \in V$ (resp. $v' \in V'$) des étiquettes $l_{(v,v')}$ telles que $v' \in V'$ (resp. $v \in V$) et à tous les arcs $e \in E$ (resp. $e' \in E'$) des étiquettes $l_{(e,e')}$ telles que $e' \in E'$ (resp. $e \in E$). Ainsi, l'étiquette $l_{(v,v')}$ d'un sommet v (resp. l'étiquette $l_{(e,e')}$ d'un arc e) est contenue dans $descr(G) \sqcap_m descr(G')$ ssi le sommet v est apparié au sommet v' (resp. l'arc e est apparié à l'arc e'). La fonction f peut donc déduire l'appariement réalisé de l'ensemble $descr(G) \sqcap_m descr(G')$ et calculer alors la même mesure que Boeres *et al.* (2004). La fonction f est définie comme une fonction de pondération associant des poids aux étiquettes $l_{(v,v')}$ (resp. $l_{(e,e')}$) en fonction de $s^v(v, v')$ (resp. $s^e(e, e')$) ou un poids très fortement négatif s'il est interdit d'apparier v à v' ou s'il existe un sommet non apparié (contrainte (i)). La fonction g permet de vérifier les contraintes (i) et (iii). Lorsqu'un appariement m associe un sommet v de l'image à plus d'un sommet (la contrainte (i) est alors violée), il existera un couple (v, s_v) dans l'ensemble $splits(m)$ auquel la fonction g pourra donner un poids infini (interdisant ainsi l'appariement m). Lorsqu'un sommet v du modèle est apparié à un ensemble s_v de sommets de l'image, le couple (v, s_v) est contenu dans $splits(m)$. La fonction g peut alors vérifier que le sous-graphe de l'image induit par les sommets de s_v est connexe et vérifier ainsi que la contrainte (iii) n'est pas violée. Notons enfin que comme le meilleur appariement au sens de notre mesure de similarité correspond au meilleur appariement au sens du problème de l'appariement non-bijectif de graphes, il est possible de calculer la mesure de Boeres *et al.* (2004) à partir de notre mesure de similarité.

Discussion

Les mesures de similarité de graphes proposées dans Ambauen *et al.* (2003) et Boeres *et al.* (2004) sont basées sur des appariement multivoques (*i.e.*, un sommet peut être apparié à plusieurs autres). Ces deux mesures ont été introduites en reconnaissance d'images pour la prise en compte des problèmes de sur-segmentation des images : les appariements multivoques permettent d'apparier une région d'une image sous-segmentée à plusieurs régions d'une image sur-segmentée.

Ces mesures sont spécifiques aux problèmes pour lesquelles elles ont été définies. Par exemple, Boeres *et al.* (2004) cherche à comparer une image réelle avec sa représentation schématique. Dans ce contexte, une région de l'image réelle doit être appariée à

une et une seule région du schéma alors qu'une région du schéma peut correspondre à plusieurs régions de l'image réelle. La mesure de similarité proposée et les algorithmes de résolution ont été construits autour de ces contraintes spécifiques et sont difficilement adaptables à un autre contexte.

A contrario, la mesure de similarité proposée dans Champin & Solnon (2003) est générique. Cette mesure de similarité est paramétrée par deux fonctions f et g qui permettent d'exprimer les contraintes et les préférences propres à un problème donné (*e.g.* la recherche d'un appariement univoque ou multivoque, l'évaluation de la qualité d'un appariement...) sont exprimées par l'intermédiaire des deux fonctions f et g. Un algorithme permettant de calculer cette similarité peut donc être utilisé dans de nombreuses applications sans effort d'adaptation. En contrepartie de cette généricité, cet algorithme sera sans doute moins efficace que des algorithmes dédiés capables d'exploiter les connaissances dépendantes du domaine d'application pour accélérer la recherche du meilleur appariement.

4 Algorithmes de mesure de similarité de graphes

Tous les problèmes d'appariement présentés en section 3 sont NP-complets ou NP-difficiles à l'exception du problème de l'isomorphisme de graphes dont la complexité n'est pas clairement établie. Pour certains graphes (tels que les arbres ou les graphes planaires) certains de ces problèmes deviennent polynomiaux (Aho *et al.* (1974); Hopcroft & Wong (1974); Luks (1982)).

Les problèmes d'isomorphismes de graphes et de sous-graphes sont généralement aisément résolus par des algorithmes complets. Des techniques très efficaces d'étiquetages des sommets ont été proposés par McKay (1981) et Sorlin & Solnon (2004) propose d'utiliser la programmation par contraintes et une méthode de filtrage ad-hoc pour le problème de l'isomorphisme de graphes. Des algorithmes complets ont été proposés pour la recherche de l'appariement qui maximise la formule (1) (Champin & Solnon (2003)) et pour le calcul de la distance étendue entre deux graphes (Ambauen *et al.* (2003)). Ces algorithmes sont basés sur une exploration exhaustive de l'espace de recherche combinée à des techniques de filtrage. Ils garantissent l'optimalité de la solution trouvée mais, du fait de l'explosion combinatoire qu'ils engendrent, ils sont limités à de très petits graphes. L'utilisation d'algorithmes incomplets qui ne garantissent pas l'optimalité de la solution trouvée mais ayant une complexité polynomiale semble être une bonne alternative. Par exemple, Boeres *et al.* (2004) propose un algorithme de construction aléatoire d'appariements non-bijectifs ainsi qu'un algorithme de recherche locale améliorant ces appariements jusqu'à l'obtention d'un maximum local. Ces deux algorithmes sont dédiés à l'appariement d'une image réelle à son modèle.

Dans cette section, nous décrivons trois algorithmes incomplets –un algorithme glouton, une recherche taboue et une recherche taboue réactive– pour le calcul de la mesure de similarité de deux graphes étiquetés. Ces algorithmes sont génériques dans le sens où ils sont paramétrés par les fonctions f et g utilisées pour introduire les connaissances et les contraintes dépendantes à une application. Ils peuvent donc être utilisés pour résoudre tout type de problèmes d'appariement de graphes.

Algorithme glouton

Cet algorithme a été proposé dans Champin & Solnon (2003). Nous le présentons brièvement car il est utilisé comme un point de départ de nos algorithmes de recherche taboue. Pour plus de précisions, nous renvoyons le lecteur à l'article original.

L'algorithme démarre d'un appariement vide $m = \emptyset$. A chaque itération, il ajoute à m un couple de sommets parmi l'ensemble $cand = V \times V' - m$ des candidats. Le couple à ajouter est choisi selon une heuristique gloutonne : le sous-ensemble des candidats dont l'ajout maximise la similarité (*i.e.*, la formule (1)) est tout d'abord construit. Ce sous-ensemble contient généralement plus d'un candidat. Afin de les départager, le "potentiel" de chaque candidat (v, v') est anticipé en prenant en compte les caractéristiques d'arcs entrants (resp. sortants) communs à v et v' et n'étant pas encore dans $descr(G) \sqcap_{m \cup \{(v,v')\}} descr(G')$. En cas d'ex-æquo, le couple de sommets à ajouter est choisi aléatoirement. Ces ajouts gloutons sont répétés jusqu'à l'obtention d'un maximum local, *i.e.*, jusqu'à ce qu'aucun couple de sommets ne puisse augmenter la similarité. Cet algorithme a une complexité en temps polynomiale de $\mathcal{O}((|V| \times |V'|)^2)$ (lorsque le calcul de f et de g est de complexité linéaire en temps). En contrepartie de cette faible complexité, l'algorithme ne fait jamais de "retour arrière" et n'est pas complet : bien qu'il puisse parfois trouver la meilleure solution, cela n'est pas systématique et on ne peut pas l'utiliser pour prouver l'éventuelle optimalité d'une solution. Cet algorithme n'étant pas déterministe, nous pouvons l'exécuter plusieurs fois et garder la meilleure solution trouvée.

Recherche locale

L'algorithme glouton retourne un appariement "localement optimal" dans le sens où il ne peut pas être amélioré en ajoutant un seul couple de sommets. Il est néanmoins possible de l'améliorer en ajoutant et en supprimant plusieurs couples de sommets. Une recherche locale tente d'améliorer une solution en explorant son voisinage. Les voisins d'un appariement m sont les appariements qui peuvent être obtenus en ajoutant ou en enlevant un couple de sommets à m.

$$\begin{aligned} \forall m \in \wp(V \times V'), voisinage(m) &= \{m \cup \{(v,v')\} | (v,v') \in (V \times V') - m\} \\ &\cup \{m - \{(v,v')\} | (v,v') \in m\} \end{aligned}$$

A partir d'un appariement initial calculé par l'algorithme glouton, l'espace de recherche est exploré de voisin en voisin jusqu'à ce que la meilleure solution soit obtenue (lorsque la qualité de celle-ci est connue) ou jusqu'à ce que le nombre d'itérations maximum soit atteint. A chaque itération, une heuristique guide la recherche en déterminant le prochain voisin à explorer.

Méta-heuristique taboue

La recherche *taboue* (Glover (1989); Dorne & Hao (1998); Petrovic *et al.* (2002)) est une des meilleures heuristiques connues pour choisir le prochain voisin à explorer.

A chaque itération, le voisin est choisi par rapport au critère de l'algorithme glouton. Notons cependant que le meilleur voisin d'un appariement m localement optimal est de moins bonne qualité que m. Afin de ne pas cantonner la recherche autour d'un maximum local en ajoutant puis en retirant continuellement le même couple de sommets, une liste taboue est utilisée. Cette liste de longueur k mémorise les k derniers mouvements effectués (*i.e.*, les k derniers couples de sommets ajoutés ou supprimés) afin d'interdire un mouvement inverse à un mouvement récemment effectué (*i.e.*, ajouter/supprimer un couple de sommets récemment supprimé/ajouté). Une exception nommée "aspiration" est ajoutée : si un mouvement interdit permet d'atteindre un appariement de meilleur qualité que le meilleur appariement connu jusqu'alors, le mouvement est quand même réalisé. La figure 1 décrit l'algorithme tabou du calcul d'une valeur approchée de la formule (2).

fonction Tabou($G = \langle V, r_V, r_E \rangle, G' = \langle V', r_{V'}, r_{E'} \rangle, k, limiteQualite, maxMouv$)
retourne un appariement $m \subseteq V \times V'$
  $m \leftarrow Glouton(G, G')$; $best_m \leftarrow m$; $nbMouv \leftarrow 0$
  tant que $sim_{best_m}(G, G') < limiteQualite$ **et** $nbMouv < maxMouv$ **faire**
    $cand \leftarrow \{m' \in voisinage(m) / sim_{m'}(G, G') > sim_{best_m}(G, G')\}$
    si $cand = \emptyset$ **alors**/* *pas d'aspiration* */
      $cand \leftarrow \{m' \in voisinage(m) / pasTabou(m, m', k)\}$
    fin si
    $cand \leftarrow \{m' \in cand / m'$ est maximal par rapport au critère de glouton$\}$
    choisir aléatoirement $m' \in cand$
    $rendreTabou(m, m', k)$; $m \leftarrow m'$; $nbMouv \leftarrow nbMouv + 1$
    si $sim_m(G, G') > sim_{best_m}(G, G')$ **alors** $best_m \leftarrow m$ **fin si**
  fin tant que
  retourner $best_m$

FIG. 1 – Algorithme Tabou

Recherche taboue réactive

La longueur k de la liste taboue est un paramètre critique et difficile à déterminer : si la liste est trop longue, la diversification de la recherche est trop forte et l'algorithme converge trop lentement ; si la liste est trop courte, l'intensification de la recherche est trop forte et l'algorithme reste autour d'un optimum local et ne trouve plus de meilleures solutions. Battiti & Protasi (2001) résolvent ce problème grâce à la recherche taboue réactive dans laquelle la longueur de la liste taboue est adaptée dynamiquement pendant la recherche. Si un même appariement est exploré plusieurs fois, la recherche doit être diversifiée. Afin de détecter une telle redondance, la clé de hachage de chaque appariement visité est mémorisée. Quand une collision a lieu dans la table de hachage, la liste taboue est allongée afin de diversifier la recherche. A contrario, quand il n'y a pas eu de collisions durant un certain nombre de mouvements (signe que la recherche est suffisamment diversifiée) la liste taboue est raccourcie. Les clés de hachage pouvant être calculées de façon incrémentale, le coût supplémentaire de ces calculs est négligeable.

5 Résultats expérimentaux

5.1 Problèmes étudiés

Nous avons expérimenté les algorithmes sur 3 types d'appariements différents :

1. Des problèmes d'isomorphisme de graphes et de sous-graphes proposés par Foggia *et al.* (2001). Les graphes ont entre 100 et 200 sommets pour les problèmes d'isomorphisme de graphes. Les problèmes d'isomorphisme de sous-graphes recherchent un sous-graphe d'un graphe ayant entre 20 et 100 sommets. Les sous-graphes ont 80%, 60% ou 40% de sommets en moins que les graphes qui les contiennent.
2. Sept instances du problème d'appariement non-bijectif de graphes proposées par Boeres *et al.* (2004). Le graphe schéma a entre 10 et 50 sommets, le graphe image a entre 30 et 250 sommets. Pour plus de détails, se référer à Boeres *et al.* (2004).
3. Une centaine d'instances du problème d'appariement multivoque de graphes. Il n'existe pas de benchmark de problèmes d'appariements multivoques. Nous avons donc conçu un générateur de paires de graphes "similaires". Un graphe est généré aléatoirement puis quelques fusions et éclatements de sommets et quelques insertions et suppressions de sommets lui sont appliqués afin d'obtenir un second graphe similaire au premier. Une borne mimimum de la similarité des deux graphes est calculée à partir de l'ensemble des transformations effectuées. Le problème est considéré résolu lorsque cette borne est atteinte. Quand les composants des graphes sont très différents de par leurs étiquettes, le meilleur appariement est facilement trouvé car le nombre de couples de sommets qu'il peut être intéressant d'apparier est faible. Afin d'obtenir des instances difficiles les graphes générés sont tels que tous les arcs et tous les sommets ont la même étiquette. Les graphes considérés ici ont 100 sommets et entre 200 et 360 arcs. Le second graphe est obtenu en fusionnant ou éclatant 5 sommets et en supprimant ou ajoutant 10 arcs ou sommets. Les fonctions f et g de la formule (2) sont des fonctions de cardinalité.

5.2 Protocole expérimental

Les trois algorithmes ont été écrits en C++ et exécutés sur un Pentium IV 2GHz avec 512Mo de RAM. Les problèmes sont modélisés en problèmes de mesure de similarité et sont tous résolus par le même programme. Une seule modification du code (facultative) a été réalisée pour le problème de Boeres *et al.* (2004). Cette modification permet d'abstraire les étiquettes afin d'accélérer le calcul de la similarité par rapport à un appariement.

Une résolution par l'algorithme glouton consiste à exécuter 500 fois l'algorithme et à retourner le meilleur appariement trouvé. Les recherches locales exécutent le même nombre de mouvements que l'algorithme glouton, *i.e.*, $500 * n$ si les appariements trouvés par l'algorithme glouton ont en moyenne n couples de sommets. En moyenne, les meilleurs résultats de la version non réactive de tabou ont été obtenus avec une liste taboue de longueur $k = 30$. Cette valeur n'étant cependant pas optimale pour toutes

les instances, les tests ont aussi été effectués pour des valeurs de k comprises entre 10 et 50. Les meilleurs paramètres trouvés pour la liste taboue réactive sont 10 (resp. 50) pour la longueur minimale *lMin* (resp. maximale *lMax*), 15 pour la longueur *lDiff* d'allongement ou de raccourcissement et 1000 mouvements pour la fréquence *freq* de raccourcissement de la liste.

5.3 Résultats

Isomorphismes. En 10 secondes, l'algorithme glouton résoud 80% des instances d'isomorphisme de graphes alors que les recherches taboues (réactives ou non) en résolvent 100%. Les problèmes d'isomorphisme de sous-graphes sont beaucoup plus difficiles : en 200s, la recherche taboue réactive résoud 66% des instances à 100 sommets alors que glouton n'en résoud que 4,4%. Ces résultats mitigés s'expliquent par le fait que notre algorithme n'utilise aucune technique de filtrage et explore donc potentiellement tous types d'appariements, même ceux qui sont multivoques.

Notons qu'une variation (même légère) du paramètre k de tabou non-réactif influence énormément les résultats et que la valeur optimale de k varie d'une instance à l'autre d'un même problème. A contrario, les paramètres de tabou réactif sont plus "robustes" : de petites variations de ces paramètres ne changent pas significativement les résultats.
Appariement non-bijectif de graphes. Le tableau de la figure 2 résume les résultats obtenus sur le problème de l'appariement non-bijectif de graphes. La colonne *Pb* donne le numéro de l'instance. La colonne *fLS+* donne la similarité maximum obtenue par Boeres *et al.* (2004), la colonne *T* (resp. *TR*) donne le résultat de tabou non réactif (resp. réactif) dans son paramétrage "standard" (voir la section 5.2). La colonne *T** donne le résultat et la longeur de la liste du meilleur tabou non réactif. La colonne *TR** donne le résultat et le paramétrage (sous la forme *lMin.lMax.lDiff.freq*) du meilleur tabou réactif. Notons que l'exécution de nos trois algorithmes sur ces instances est déterministe : les poids manipulés par ces instances sont des réels et il n'y a jamais eu d'ex-æquo. Aucun couple n'est donc choisi aléatoirement lors de l'exécution de ces instances.

Pb	fLS+	TR	TR*	T	T*
5	.5474	.5481	.5548(10.50.15.750)	.5463	.5463(30)
5a	.5435	.5529	.5597(10.50.15.750)	.5519	.5558(20)
6	.4248	.4213	.4213(Tous)	.4213	.4213(Tous)
7	.6319	.6333	.6354(10.50.15.500)	.6342	.6344(35)
8	.5186	.5210	.5212(10.50.10.500)	.5195	.5204(45)
8a	.5222	.5245	.5248(10.50.10.1000)	.5231	.5240(50)
9	.5187	.5199	.5202(15.100.20.750)	.5198	.5198(50)

FIG. 2 – Problème d'appariements non-bijectifs

Pour 5 instances sur 7, tabou non réactif obtient de meilleurs résultats que la recherche locale de Boeres *et al.* (2004). Toute instance confondue, les meilleurs résultats *en moyenne* sont obtenus avec une longueur de liste k à 30. Cependant, les informations de la colonne *T** montre que ce paramétrage ne permet d'obtenir le meilleur résultat par instance que pour 2 instances sur 7 : la longueur optimale de la liste taboue varie entre 20 et 50 selon les instances. Un paramétrage standard de tabou réactif permet d'obtenir

de meilleurs résultats que Boeres *et al.* (2004) et que tabou non réactif pour 6 instances sur 7. Ces résultats peuvent encore être améliorés avec un paramétrage fin (colonne *TR**). Seule l'instance 6 pose problème à nos algorithmes (aucune de nos recherches locales n'a réussi à améliorer l'appariement proposé par glouton). La généricité et l'efficacité de notre approche a un prix : les temps d'exécution de nos recherches locales sont souvent supérieurs à ceux de Boeres *et al.* (2004). Bien que la comparaison des temps d'exécution soit difficile (les machines utilisées n'étant pas les mêmes), le temps d'exécution moyen de tabou réactif est de 20s environ contre 11,7s pour l'algorithme de Boeres *et al.* (2004).

Appariements multivoques. Chaque algorithme a été exécuté 200 fois sur chacune des 100 instances générées. 51% des intances se sont avérées "faciles" dans le sens où elles sont toujours résolues par l'algorithme glouton. Sur les 49 instances restantes, 35 ont été facilement –et tout le temps– résolues par chacune des recherches locales (moins de 500 mouvements exécutés en moins de 4s). Les 14 dernières instances se sont avérées beaucoup plus difficiles et nécessitent plus de 25000 mouvements pour être résolues. La version non réactive de tabou réussit dans 64% des exécutions alors que la version réactive obtient un succès dans 79% des exécutions. En outre, la version réactive de tabou semble plus robuste que la version non réactive : le choix des paramètres a une moindre influence sur les résultats. La version réactive est donc plus efficace et plus facilement paramétrable que la version non réactive.

6 Conclusion

Nous avons montré que la mesure de similarité de Champin & Solnon (2003) est plus générique que les autres mesures de similarité de graphes existantes (Bunke (2000); Conte *et al.* (2004)). Cette mesure est basée sur des appariements multivoques des sommets des graphes ce qui permet de prendre en compte les problèmes de granularité (en représentation de connaissances) ou les problèmes de sur- et sous-segmentation des images en reconnaisssance d'images (Ambauen *et al.* (2003); Boeres *et al.* (2004)). Nous présentons trois algorithmes de complexité polynomiale : un algorithme glouton, une recherche locale basée sur la méta-heuristique taboue et une version améliorée de cette dernière nommée "recherche taboue réactive". Nous montrons l'efficacité de notre approche sur trois types de problèmes et en particulier sur le problème proposé par Boeres *et al.* (2004).

La mesure de similarité de Champin & Solnon (2003) est définie pour des graphes multi-étiquetés : à chaque sommet et chaque arc est associé un ensemble d'étiquettes décrivant leurs propriétés. Le multi-étiquetage des graphes permet une description très fine des objets. Par exemple, dans un contexte de représentation d'images segmentées, le sommet d'un graphe peut être associé à une étiquette décrivant la couleur de la région de l'image correspondante, une autre étiquette décrivant sa taille, une autre décrivant sa forme... Lors de l'appariement de deux images, il est alors possible de déterminer d'une part à quel point les images sont similaires (en comptant le nombre d'étiquettes retrouvées) et d'autre part en quoi elles sont similaires (en regardant précisement quelles étiquettes ont été retrouvées).

Notre mesure générique, associée au pouvoir d'expression des graphes multi-étiquetés

pourrait être utilisée pour définir une nouvelle mesure de similarité d'images. Nous envisageons donc maintenant de nous associer à des spécialistes de la reconnaissance d'image afin de proposer de nouvelles méthodes de recherche et de classification d'images.

Références

AHO A., HOPCROFT J. & ULLMAN J. (1974). *The design and analysis of computer algorithms*. Addison Wesley.

AMBAUEN R., FISCHER S. & BUNKE H. (2003). Graph Edit Distance with Node Splitting and Merging, and Its Application to Diatom Identification. In *IAPR-TC15 Wksp on Graph-based Representation in Pattern Recognition*, p. 95–106.

BATTITI R. & PROTASI M. (2001). Reactive local search for the maximum clique problem. In SPRINGER-VERLAG, Ed., *Algorithmica*, volume 29, p. 610–637.

BOERES M., RIBEIRO C. & BLOCH I. (2004). A randomized heuristic for scene recognition by graph matching. In *WEA 2004*, p. 100–113.

BUNKE H. (1997). On a relation between graph edit distance and maximum common subgraph. *PRL : Pattern Recognition Letters*, **18**.

BUNKE H. (2000). Graph matching : Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface 2000, Montreal*, p. 82–88.

CHAMPIN P.-A. & SOLNON C. (2003). Measuring the similarity of labeled graphs. In *5th International Conference on Case-Based Reasoning (ICCBR 2003)*, volume Lecture Notes in Artificial Intelligence 2689-Springer-Verlag, p. 80–95.

CONTE D., FOGGIA P., SANSONE C. & VENTO M. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, **18**(3), 265–298.

DORNE R. & HAO J. (1998). *Tabu Search for graph coloring, T-coloring and Set T-colorings*, In *Metaheuristics 98 : Theory and Applications*, chapter 3. I.H. Osman et al. (Eds.), Kluver Academic Publishers.

FOGGIA P., SANSONE C. & VENTO M. (2001). A database of graphs for isomorphism and sub-graph isomorphism benchmarking. *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, p. 176 –187.

GLOVER F. (1989). Tabu search - part I. *Journal on Computing*, p. 190–260.

HOPCROFT J. & WONG J. (1974). Linear time algorithm for isomorphism of planar graphs. 6^{th} *Annu. ACM Symp. theory of Comput.*, p. 172–184.

LUKS E. (1982). Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer System Science*, p. 42–65.

MCKAY B. (1981). Practical graph isomorphism. *Congressus Numerantium*, **30**, 45–87.

PETROVIC S., KENDALL G. & YANG Y. (2002). A Tabu Search Approach for Graph-Structured Case Retrieval. In *STAIRS 2002*, p. 55–64.

SORLIN S. & SOLNON C. (2004). A global constraint for graph isomorphism problems. In *the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2004)*.

Les arbres aux feuilles ordonnées : une théorie complète pour la représentation des connaissances

Khalil Djelloul

Laboratoire d'Informatique Fondamentale de Marseille.
Parc scientifique et technologique de Luminy.
163 avenue de Luminy - Case 901.
13288 Marseille, cedex 9. France.
khalil.djelloul@lif.univ-mrs.fr
http://www.lif.univ-mrs.fr/∼djelloul

Résumé :

Nous établissons tout d'abord un théorème général qui donne une condition suffisante pour qu'une théorie du 1er ordre soit complète. Ce théorème fait intervenir un quantificateur spécial, affirmant essentiellement l'existence d'une infinité d'individus ayant une propriété du premier ordre donnée. La démonstration du théorème n'est rien d'autre que les grandes lignes d'un algorithme général qui décide si une proposition ou sa négation est vraie dans certaines théories.

Nous introduisons ensuite une nouvelle théorie T_A dédiée à la représentation des connaissances et à la résolution de contraintes quantifiées faisant intervenir des arbres et des numériques ordonnés. Cette théorie a pour modèle un ensemble d'arbres munis d'opérations de construction et d'un ordre total dense sans extrêmes portant sur les feuilles de ces arbres. A l'aide du théorème nous montrons alors que T_A est une théorie complète. De notre démonstration il est possible d'extraire un algorithme général de résolution de contraintes quantifiées dans T_A.

Mots-clés : Théorie complète, Représentation des connaissances, Arbres, Ordre dense, Résolution de contraintes dans des structures hybrides.

1 Introduction

Rappelons qu'un arbre construit sur un ensemble E est fondamentalement un ensemble hiérarchisé de nœuds étiquetés par les éléments de E. A chaque élément e de E est associé une opération f, dite *de construction*, qui, à partir d'une suite $a_1, \ldots, a_n$ d'arbres, construit l'arbre dont la racine est étiquetée e et dont la suite des fils immédiats est $a_1, \ldots, a_n$.

Les arbres éventuellement infinis jouent un rôle fondamental en programmation et en représentation des connaissances. Ils modélisent les données composées comme on

les connaît sous le nom *d'enregistrement* ou de *record* en Pascal ou de *structure* en C. L'opération de construction correspond à la création d'un nouvel enregistrement, c'est-à-dire d'une cellule contenant une information élémentaire suivie éventuellement de n cellules, chacune contenant un pointeur vers un enregistrement. Il y a création d'arbre infini lors de la création d'un circuit de pointeurs.

Dès 1976, Gérard Huet a proposé un algorithme pour unifier des termes infinis, c'est-à-dire, résoudre des équations dans les arbres (Huet, 1976). Bruno Courcelle a étudié les propriétés des arbres infinis notamment dans le cadre des schémas récursif de programme (Courcelle, 1983; Courcelle, 1986). Alain Colmerauer a modélisé le fonctionnement de Prolog II, III et IV par la résolution d'équations et de diséquations dans les arbres infinis (Colmerauer, 1984). Michael Maher a donné et justifié des axiomatisations complètes de différents ensembles d'arbres munis d'opérations de construction (Maher, 1988). Pour chacun de ces ensembles, il a donc proposé une théorie complète, c'est-à-dire un ensemble de propriétés (écrites sous forme de propositions) du 1er ordre à partir desquelles on peut déduire toutes les autres propriétés du 1er ordre de cet ensemble.

Pour notre part nous établissons ici un théorème général qui donne une condition suffisante pour qu'une théorie du premier ordre soit complète et nous l'utilisons pour montrer la complétude de l'ensemble des arbres, éventuellement infinis, construits sur un ensemble infini E muni non seulement d'opérations de construction mais aussi d'une relation d'ordre total (dense et sans extrémités) portant sur les feuilles de ces arbres. Cette nouvelle théorie est dotée d'un grand pouvoir d'expression et permet ainsi de modéliser et de représenter des problèmes complexes par des formules du premier ordre. La solution du problème n'est rien d'autre que la formule finale résolue dans cette théorie.

Le papier est organisé en quatre sections suivies d'une conclusion. Cette introduction constitue la première section. La deuxième introduit les notions de logique du premier ordre dont nous aurons besoin et se termine par la présentation d'une condition suffisante fondamentale pour qu'une théorie soit complète. C'est une variante de celle donnée à la section 1.5 de (Chang & Keisler, 1988).

La troisième section est consacrée à une condition suffisante, beaucoup plus élaborée, pour qu'une théorie soit complète. L'idée de base de cette condition provient d'un algorithme de décomposition introduit par Thi-Bich-Hanh Dao dans sa thèse (Hanh, 2000). Elle consiste à décomposer une suite de quantifications existentielles portant sur une conjonction de formules, en trois suites imbriquées de quantifications ayant des propriétés très particulières, exprimables à l'aide de trois quantificateurs spéciaux notés $\exists?$, $\exists!$, $\exists_o^{\Psi(u)}{}_\infty$ et appelés *au-plus-un*, *un-et-un-seul*, *zéro-infini*. Ces quantificateurs spéciaux ainsi que leurs propriétés sont décrits en début de section. Alors que les quantificateurs $\exists?$, $\exists!$ ne sont que de simples notations commodes, le quantificateur $\exists_o^{\Psi(u)}{}_\infty$, une des contributions essentielles de ce papier, exprime une propriété non exprimable au premier ordre.

La quatrième section introduit la théorie T_A des arbres aux feuilles ordonnées ainsi que son modèle standard A. On y étudie aussi des formules particulières appelées *briques* car elles servent à construire toutes les formules dont nous avons besoin. On terminera cette section par l'énoncé de la complétude de T_A.

Par manque de place, nous avons jugé utile de présenter une version complète de notre

article à l'adresse http ://www.lif.univ-mrs.fr/∼djelloul. On y trouvera les démonstrations de toutes les propriétés énnoncées ansi que la démonstration de la complétude de T_A en utilisant le théorème établi à la troisième section. Cette démonstration ainsi que notre théorème général sont les deux grands résultats de ce papier.

2 Rappels sur la logique du premier ordre

2.1 Expressions

On se donne, une fois pour toutes, un ensemble infini dénombrable **V** de *variables* et l'ensemble L de symboles *logiques* :

$$=, vrai, faux, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists, (,).$$

On se donne aussi, une fois pour toutes, une *signature* S, c'est-à-dire un ensemble de symboles partitionné en deux sous-ensembles : l'ensemble des symboles *de fonction* et l'ensemble des symboles *de relation*. A chaque symbole de fonction et de relation est attaché un entier n positif ou nul, son *arité*. Un symbole n-aire est un symbole d'arité n. Un symbole de fonction 0-aire est appelé *constante*.

Une *expression*, bien entendu, est un mot sur $L \cup S$ qui est soit un *terme*, c'est-à-dire un mot de l'une des deux formes :

$$x,\ ft_1 \dots t_n, \tag{1}$$

soit une *formule*, c'est-à-dire un mot de l'une des onze formes :

$$\begin{array}{c} s = t,\ rt_1 \dots t_n,\ vrai,\ faux, \\ \neg\varphi,\ (\varphi \wedge \psi),\ (\varphi \vee \psi),\ (\varphi \rightarrow \psi),\ (\varphi \leftrightarrow \psi), \\ (\forall x\, \varphi),\ (\exists x\, \varphi), \end{array} \tag{2}$$

avec dans (1), x pris dans **V**, f un symbole de fonction n-aire pris dans S et t_i des termes de tailles plus petites que celui qui est en train d'être défini, et avec dans (2), s, t, t_i des termes, r un symbole de relation n-aire pris dans S et φ et ψ des formules de tailles plus petites que celles qui sont en train d'être définies.

Les formules de la première ligne de (2) sont dites *atomiques*, et *à plat* si elles sont de l'une des formes :

$$vrai,\ faux,\ x_0 = x_1,\ x_0 = fx_1 ... x_n,\ rx_1 ... x_n,$$

où les x_i sont des variables prises dans **V**.

On rappelle qu'une occurrence d'une variable x dans une formule est *liée* si elle se produit à l'intérieur d'une sous-formule de la forme $(\forall x\, \varphi)$ ou $(\exists x\, \varphi)$. Elle est *libre* dans le cas contraire. Les *variables libres d'une formule* sont celles qui ont au moins une occurrence libre dans cette formule. Une *proposition* est une formule sans variables libres.

Nous ne distinguons pas deux formules qui peuvent être rendues égales moyennant les transformations suivantes des sous-formules :

$$(\varphi \wedge \psi) \wedge \phi \Longrightarrow \varphi \wedge (\psi \wedge \phi),\ \varphi \wedge \psi \Longrightarrow \psi \wedge \varphi,$$
$$\varphi \wedge \mathit{vrai} \Longrightarrow \varphi,\ \varphi \vee \mathit{faux} \Longrightarrow \varphi.$$

Si I est l'ensemble $\{i_1, ..., i_n\}$, nous écrivons respectivement $\bigwedge_{i\in I} \varphi_i$ et $\bigvee_{i\in I} \varphi_i$ pour $\varphi_{i_1} \wedge \varphi_{i_2} \wedge ... \wedge \varphi_{i_n} \wedge \mathit{vrai}$, et $\varphi_{i_1} \vee \varphi_{i_2} \vee ... \vee \varphi_{i_n} \vee \mathit{faux}$, . En particulier pour $I = \emptyset$, les formules $\bigwedge_{i\in I} \varphi_i$ et $\bigvee_{i\in I} \varphi_i$ se réduisent respectivement à vrai et à faux. On écrit $Card(I)$ pour désigner la cardinalité de cet ensemble I.

La syntaxe des expressions étant contraignante, on se permet d'utiliser des notations infixées pour les symboles binaires, d'ajouter des parenthèses et d'en enlever quand il n'y a pas d'ambiguïtés.

2.2 Modèle

Un *modèle*, est un couple $M = (\mathcal{M}, \mathcal{F})$, où $\mathcal{F}$ est une famille d'opérations et de relations dans l'ensemble $\mathcal{M}$, indicées par les éléments de S. Plus précisément si $\mathcal{F}$ est notée $(s^M)_{s\in S}$, alors :

- $\mathcal{M}$, *l'univers* ou *domaine* de M, est un ensemble non vide disjoint de S ; ses éléments sont appelés *individus* de M ;
- pour chaque symbole f de fonction n-aire pris dans S, f^M est une opération n-aire dans $\mathcal{M}$, c'est-à-dire une application de $\mathcal{M}^n$ dans $\mathcal{M}$; en particulier lorsque f est une constante f^M est un élément de $\mathcal{M}$;
- pour chaque symbole r de relation n-aire pris dans S, r^M est une relation n-aire dans $\mathcal{M}$, c'est-à-dire un sous-ensemble de $\mathcal{M}^n$.

Si M est un modèle de domaine $\mathcal{M}$, une *M-expression* φ est une expression construite sur la signature $S \cup \mathcal{M}$ au lieu de S en considérant les éléments de $\mathcal{M}$ comme des symboles de fonction d'arité 0. Si pour toute variable libre x de φ, on remplace chaque occurrence libre de x par un même élément de $\mathcal{M}$, on obtient une M-expression appelée *instanciation* de φ par des individus de M.

Si φ est une M-formule, on dit que φ *est vraie dans* M et on écrit

$$M \models \varphi, \tag{3}$$

pour signifier que, pour toute instanciation φ' de φ par des individus de M, l'ensemble $\mathcal{M}$ a la propriété exprimée par $\varphi\prime$, lorsqu'on interprète les symboles de fonction et de relation de $\varphi\prime$ par les fonctions et relations correspondantes de M, et lorsqu'on attribue aux symboles logiques leur sens habituel.

Remarque 2.2.1
Pour toute M-formule φ sans variables libres, une et une seule des propriétés suivantes est vérifiée : $M \models \varphi$, $M \models \neg\varphi$.

Terminons cette sous-section par une notation commode. Soit $\bar{x} = x_1...x_n$ un mot sur **V** et soit $\bar{i} = i_1...i_n$ un mot sur $\mathcal{M}$ ou **V** de la même taille que $\bar{x}$. Si on désigne par $\varphi(\bar{x})$ une M-formule, alors on désignera par $\varphi(\bar{i})$ la M-formule obtenue en remplaçant chaque occurrence libre de x_j par i_j.

2.3 Théorie

Une *théorie*, est un ensemble (éventuellement infini) de formules sans variables libres, appelées *axiomes*. On dit que le modèle M est *un modèle de* T, si $M \models \varphi$, pour tout élément φ de T. Si φ est une formule, on écrit $T \models \varphi$, pour signifier que $M \models \varphi$ pour tout modèle M de T. On dit que les formules φ et ψ sont *équivalentes dans* T ssi $T \models \varphi \leftrightarrow \psi$.

2.4 Théorie complète

Dans ce qui suit nous utilisons l'abréviation svls pour ¡¡ sans variables libres supplémentaires ¿¿.

Définition 2.4.1
Une théorie T *est* complète *si, pour toute proposition* φ*, une et une seule des propriétés suivantes est vérifiée :* $T \models \varphi$, $T \models \neg\varphi$.

Propriété 2.4.2
Une condition suffisante pour qu'une théorie T *soit complète est qu'il existe un ensemble de formules, dites* de base*, tel que :*

1. *toute formule atomique à plat soit équivalente, dans* T*, à une combinaison booléenne de formules de base svls,*
2. *toute formule de base sans variables libres soit équivalente, soit à* $vrai$*, soit à* $faux$*, dans* T*,*
3. *toute formule de la forme*

$$\exists x\,(\textstyle\bigwedge_{i\in I}\varphi_i) \wedge (\bigwedge_{i\in I'}\neg\varphi_i), \tag{4}$$

avec les φ_i *des formules de base, soit équivalente, dans* T*, à une combinaison booléenne de formules de base svls.*

3 Un théorème de complétude de théorie

3.1 Quantificateurs vectoriels

Soit M un modèle et T une théorie, tous les deux de signature S. Soient $\bar{x} = x_1 \dots x_n$ et $\bar{y} = y_1 \dots y_n$ deux mots sur $\mathbf{V}$ de même taille et soient ψ, ϕ, φ et $\varphi(\bar{x})$ des M-formules.

Notation 3.1.1
On écrit

$\exists\bar{x}\,\varphi$	*pour* $\exists x_1 ... \exists x_n\,\varphi$,
$\forall\bar{x}\,\varphi$	*pour* $\forall x_1 ... \forall x_n\,\varphi$,
$\exists?\bar{x}\,\varphi(\bar{x})$	*pour* $\forall\bar{x}\forall\bar{y}\,\varphi(\bar{x}) \wedge \varphi(\bar{y}) \rightarrow \bigwedge_{i\in\{1,\dots,n\}} x_i = y_i$,
$\exists!\bar{x}\,\varphi$	*pour* $(\exists\bar{x}\,\varphi) \wedge (\exists?\bar{x}\,\varphi)$.

Le mot $\bar{x}$, qui peut être le mot vide ε, est appelé *vecteur de variables*. On remarque que les formules $\exists ?\varepsilon\varphi$ et $\exists !\varepsilon\varphi$ sont respectivement équivalentes à $vrai$ et φ dans tout modèle.

Propriété 3.1.2
Si $T \models \exists ?\bar{x}\,\varphi$ *alors*

$$T \models (\exists \bar{x}\,\varphi \wedge \neg\phi) \leftrightarrow (\exists \bar{x}\varphi) \wedge \neg(\exists \bar{x}\,\varphi \wedge \phi).$$

Propriété 3.1.3
Si $T \models \exists ?\bar{y}\phi$ *et si aucune variable de* $\bar{y}$ *n'a d'occurrence libre dans* φ *alors*

$$T \models (\exists \bar{x}\,\varphi \wedge \neg(\exists \bar{y}\,\phi \wedge \psi)) \leftrightarrow \begin{bmatrix}(\exists \bar{x}\,\varphi \wedge \neg(\exists \bar{y}\,\phi))\vee \\ (\exists \bar{x}\bar{y}\,\varphi \wedge \phi \wedge \neg\psi)\end{bmatrix}.$$

Propriété 3.1.4
Si $T \models \psi \rightarrow (\exists !\bar{x}\,\varphi)$ *alors*

$$T \models (\psi \wedge (\exists \bar{x}\,\varphi \wedge \neg\phi)) \leftrightarrow (\psi \wedge \neg(\exists \bar{x}\,\varphi \wedge \phi)).$$

3.2 Quantificateur zéro-infini

Soit M un modèle et T une théorie, tous les deux de signature S. Soient φ_i, et $\varphi(\bar{x})$ deux M-formules et soit $\Psi(u)$ un ensemble de formules ayant au plus u comme variable libre.

Définition 3.2.1
On écrit

$$M \models \exists_o^{\Psi(u)}{}_{\infty} x\,\varphi(x), \tag{5}$$

si pour toute instanciation $\exists x\,\varphi'(x)$ *de* $\exists x\,\varphi(x)$ *par des individus de* M *:*
- *soit l'ensemble des individus* i *de* M *tel que* $M \models \varphi'(i)$ *est vide,*
- *soit, pour tout ensemble fini* $\{\psi_1(u), .., \psi_n(u)\}$ *d'éléments de* $\Psi(u)$*, l'ensemble des individus* i *de* M *tels que* $M \models \varphi'(i) \wedge \bigwedge_{j\in\{1,...,n\}} \neg\psi_j(i)$ *est infini.*

On écrit

$$T \models \exists_o^{\Psi(u)}{}_{\infty} x\,\varphi(x),$$

si pour tout modèle M *de* T *on a (5).*

Propriété 3.2.2
Une condition suffisante pour que

$$T \models (\exists x\,\varphi(x) \wedge \textstyle\bigwedge_{i\in I} \neg\varphi_i) \leftrightarrow (\exists x\,\varphi(x))$$

est que $T \models \exists_o^{\Psi(u)}{}_{\infty} x\,\varphi(x)$ *et pour chaque* φ_i*, une au moins des propriétés suivantes est vérifiée :*
- $T \models \exists ?x\,\varphi_i$*,*
- *il existe* $\psi_i(u) \in \Psi(u)$ *tel que* $T \models \forall x\,\varphi_i \rightarrow \psi_i(x)$.

3.3 Le théorème de complétude

Théorème 3.3.1
Une condition suffisante pour qu'une théorie T soit complète est qu'il existe un ensemble de formules $\psi(u)$, ayant au plus u comme variable libre, un ensemble A de formules, fermé pour la conjonction et le renommage, un ensemble A' de formules de la forme $\exists\bar{x}\alpha$, avec $\alpha \in A$, et un sous-ensemble A'' de A tels que :

1. *toute formule atomique à plat soit équivalente, dans T, à une combinaison booléenne, svls, de formules de la forme $\exists\bar{x}\alpha$ avec $\alpha \in A$,*
2. *toute formule sans variables libres de la forme $\exists\bar{x}'\alpha' \wedge \alpha''$ avec $\exists\bar{x}'\alpha' \in A'$ et $\alpha'' \in A''$ soit équivalente à $vrai$ dans T,*
3. *toute formule de la forme $\exists\bar{x}\,\alpha \wedge \psi$, avec α pris dans A et ψ une formule quelconque, soit équivalente, dans T, à $faux$ ou à une formule svls de la forme :*

   $$\exists\bar{x}'\,\alpha' \wedge (\exists\bar{x}''\,\alpha'' \wedge (\exists\bar{x}'''\,\alpha''' \wedge \psi)),$$

   avec $\exists\bar{x}'\,\alpha' \in A'$, $\alpha'' \in A''$, $\alpha''' \in A$ et $T \models \forall\bar{x}''\alpha'' \rightarrow \exists!\bar{x}'''\alpha'''$,
4. *si $\exists\bar{x}'\alpha' \in A'$ alors $T \models \exists?\bar{x}'\,\alpha'$ et pour chacune de ses variables libres y, une et au moins une des propriétés suivantes soit vérifiée :*
   - *$T \models \exists?y\bar{x}'\,\alpha'$,*
   - *il existe $\psi(u) \in \Psi(u)$ tel que $T \models \forall y\,(\exists\bar{x}'\,\alpha') \rightarrow \psi(y)$,*
5. *si $\alpha'' \in A''$ alors*
   - *la formule $\neg\alpha''$ soit équivalente dans T à une formule svls de la forme $\bigvee_{i\in I}\alpha_i$ avec $\alpha_i \in A$,*
   - *pour tout x'', la formule $\exists x''\alpha''$ soit équivalente, dans T, à une formule svls élément de A'',*
   - *pour tout x'', on ait $T \models \exists_{o\,\infty}^{\Psi(u)}x''\,\alpha''$.*

Faisons d'abord quelque remarques sur les propriétés engendrées par les cinq conditions du théorème 3.3.1. Si T est une théorie qui satisfait aux cinq conditions du théorème 3.3.1 alors :

Propriété 3.3.2
Toute formule de la forme $\exists\bar{x}\,\alpha$, avec $\alpha \in A$, est équivalente, dans T, à faux ou à une formule svls de la forme $\exists\bar{x}'\,\alpha' \wedge \alpha''$, avec $\exists\bar{x}'\alpha' \in A'$ et $\alpha'' \in A''$.

Propriété 3.3.3
Toute formule de la forme

$$\exists\bar{x}\,\alpha \wedge \textstyle\bigwedge_{i\in I}\neg(\exists\bar{y}_i\beta_i),$$

avec $\alpha \in A$ et $\beta_i \in A$, est équivalente dans T à faux ou à une formule svls de la forme

$$\exists\bar{x}'\,\alpha' \wedge (\exists\bar{x}''\,\alpha'' \wedge \textstyle\bigwedge_{j\in J}\neg(\exists\bar{y}'_j\,\beta'_j \wedge \beta''_j)),$$

avec $\exists\bar{x}'\alpha' \in A'$, $\alpha'' \in A''$, $\exists\bar{y}'_j\beta'_j \in A'$, $\beta''_j \in A''$ et $Card(I) = Card(J)$.

Corollaire 3.3.4
Toute formule de la forme

$$\exists \bar{x}\, \alpha \wedge \bigwedge_{i\in I} \neg(\exists \bar{y}_i \beta_i),$$

avec $\alpha \in A$ et $\beta_i \in A$, est équivalente dans T à faux ou à une disjonction de formules, svls, de la forme

$$\exists \bar{x}'\, \alpha' \wedge (\exists \bar{x}''\, \alpha'' \wedge \bigwedge_{j\in J} \neg(\exists \bar{y}'_j\, \beta'_j)),$$

avec $\exists \bar{x}'\alpha' \in A'$, $\alpha'' \in A''$ et $\exists \bar{y}'_j\, \beta'_j \in A'$.

Les Propriétés 3.3.2 et 3.3.3 se démontrent en utilisant la Propriété 3.1.4 ainsi que les points 3 et 5 du Théorème 3.3.1. Le Corollaire 3.3.4 se démontre en utilisant plusieurs fois les Propriétés 3.3.3 et 3.1.3 ainsi que les points 4 et 5 du Théorème 3.3.1.
Preuve du Théorème 3.3.1 Supposons que T soit une théorie respectant les cinq conditions du Théorème 3.3.1. Montrons que T est une théorie complète en utilisant la Propriété 2.4.2 et en prenant comme ensemble de formules de base les formules de la forme $\exists \bar{x}\alpha$ avec $\alpha \in A$.

Montrons que le point 1 de la Propriété 2.4.2 est vérifié. Si φ est une formule atomique à plat, alors, d'après le point 1 du Théorème 3.3.1, φ est équivalente dans T à une combinaison booléenne de formules de base svls.

Montrons que le point 2 de la Propriété 2.4.2 est vérifié. Si φ est une formule de base sans variables libres, alors, d'après la Propriété 3.3.2 et le point 2 du Théorème 3.3.1, φ est équivalente dans T, soit à $vrai$, soit à $faux$.

Montrons que le point 3 de la Propriété 2.4.2 est vérifié. Soit alors une formule de la forme

$$\exists x\, (\bigwedge_{i\in I} (\exists \bar{x}_i\, \alpha_i)) \wedge (\bigwedge_{j\in J} \neg(\exists \bar{y}_j\, \beta_j)),$$

avec les α_i et β_j des éléments de A. Il faut montrer que cette formule est équivalente dans T à une combinaison booléenne de formules de base svls. En remontant les quantifications $\exists \bar{x}_i$, après avoir éventuellement renommé certaines variables des $\bar{x}_i$, on obtient une formule svls, équivalente dans T, de la forme

$$\exists \bar{x}\, \alpha \wedge \bigwedge_{j\in J} \neg(\exists \bar{y}_j \beta_j),$$

avec α et β_j éléments de A car A est fermé pour la conjonction et le renommage. D'après le Corollaire 3.3.4, la formule précédente est équivalente dans T à une disjonction de formules svls de la forme

$$\exists \bar{x}'\, \alpha' \wedge (\exists \bar{x}''\alpha'' \wedge \bigwedge_{i\in I} \neg(\exists \bar{y}'_i \beta'_i)). \tag{6}$$

avec $\exists \bar{x}'\, \alpha' \in A'$, $\alpha'' \in A''$ et $\exists \bar{y}'_i \beta'_i \in A'$. Montrons maintenant que chaque formule de cette disjonction est équivalente, dans T, à une combinaison booléenne, svls, de formules de base. Soit φ une formule de la forme (6). En désignant par I_1 l'ensemble des $i \in I$ tels que x''_n n'ait pas d'occurrences libres dans $\exists \bar{y}'_i \beta'_i$, φ est équivalente dans T, à la formule svls

$$\exists \bar{x}'\alpha' \wedge (\exists x''_1 ... \exists x''_{n-1} \begin{bmatrix} (\bigwedge_{i\in I_1} \neg(\exists \bar{y}'_i \beta'_i)) \wedge \\ (\exists x''_n\, \alpha'' \wedge \bigwedge_{i\in I-I_1} \neg(\exists \bar{y}'_i \beta'_i)) \end{bmatrix}),$$

qui du fait que $\alpha'' \in A''$ et $\exists \bar{y}'_i \beta'_i \in A'$ et du fait de la Propriété 3.2.2 et des points 4 et 5 du Théorème 3.3.1, est équivalent dans T, à la formule svls

$$\exists \bar{x}' \alpha' \wedge (\exists x''_1 ... \exists x''_{n-1} (\bigwedge_{i \in I_1} \neg(\exists \bar{y}'_i \beta'_i)) \wedge (\exists x''_n \alpha'')),$$

qui du fait que $\alpha'' \in A''$ et du fait de la deuxième condition du point 5 du Théorème 3.3.1, est équivalente dans T à une formule svls de la forme

$$\exists \bar{x}' \alpha' \wedge (\exists x''_1 ... \exists x''_{n-1} (\bigwedge_{i \in I_1} \neg(\exists \bar{y}'_i \beta'_i)) \wedge \alpha''_n),$$

avec $\exists \bar{x}' \alpha' \in A'$, $\alpha''_n \in A''$ et $\exists \bar{y}'_i \beta'_i \in A'$. C'est-à-dire à

$$\exists \bar{x}' \alpha' \wedge (\exists x''_1 ... \exists x''_{n-1} \alpha''_n \wedge \bigwedge_{i \in I_1} \neg(\exists \bar{y}'_i \beta'_i)).$$

En répétant les trois étapes précédentes $n-1$ fois et en désignant par I_k l'ensemble des $i \in I_{k-1}$ tels que $x''_{(n-k+1)}$ n'ait pas d'occurrences libres dans $\exists \bar{y}'_i \beta'_i$, on obtient une formule svls, équivalente dans T, de la forme

$$\exists \bar{x}' \alpha' \wedge \alpha''_1 \wedge \bigwedge_{i \in I_n} \neg(\exists \bar{y}'_i \beta'_i),$$

avec $\exists \bar{x}' \, \alpha' \in A'$, $\alpha''_1 \in A''$ et $\exists \bar{y}'_i \beta'_i \in A'$. Du fait que $\exists \bar{x}' \alpha' \in A'$, d'après le point 4 du Théorème 3.3.1, on a $T \models \exists ? \bar{x}' \alpha'$ et donc $T \models \exists ? \bar{x}' \alpha' \wedge \alpha''_1$. D'après la Propriété 3.1.2, la formule précédente est alors équivalente, dans T, à la formule svls

$$(\exists \bar{x}' \alpha' \wedge \alpha''_1) \wedge \bigwedge_{i \in I_n} \neg(\exists \bar{x}' \, \alpha' \wedge \alpha''_1 \wedge \exists \bar{y}'_i \beta'_i),$$

qui en remontant les quantifications $\exists \bar{y}'_i$ en renommant éventuellement certaines variables des $\bar{y}'_i$ est équivalente, dans T, à une formule svls de la forme

$$(\exists \bar{x}' \alpha' \wedge \alpha''_1) \wedge \bigwedge_{i \in I_n} \neg(\exists \bar{x}' \bar{y}'_i \, \alpha' \wedge \alpha''_1 \wedge \beta'_i),$$

avec α', α''_1, β'_i des éléments de A, car A est fermé pour le renommage et A'' est un sous-ensemble de A. Comme les formules α', α''_1, β'_i sont des éléments de A et que A est fermé pour la conjonction, la formule précédente est bien une combinaison booléenne de formules de base. Le point 3 de la Propriété 2.4.2 est donc vérifié et par conséquent le Théorème 3.3.1 est démontré.

4 Théorie T_A des arbres aux feuilles ordonnées

4.1 Présentation de T_A

La signature de T_A est composée d'un ensemble infini **F** de symboles de fonctions d'arités non nulles et des symboles de relations *feuille*, *arbre*, $<$ d'arité respective 1,1,2. Si t_1 et t_2 sont des termes, on écrit $t_1 < t_2$ pour $< (t_1, t_2)$. Les axiomes de T_A sont les

propositions de l'une des treize formes suivantes :

1 $\forall \bar{x} \forall \bar{y}\, f\bar{x} = f\bar{y} \rightarrow \bigwedge_i x_i = y_i$,
2 $\forall \bar{x} \forall \bar{y}\, \neg f\bar{x} = g\bar{y}$,
3 $\forall \bar{x} \exists! \bar{y}\, \bigwedge_i y_i = t_i(\bar{y}, \bar{x})$,
4 $\forall x\, feuille\, x \rightarrow \neg x < x$,
5 $\forall x \forall y \forall z\, feuille\, x \wedge feuille\, y \wedge feuille\, z \rightarrow ((x < y \wedge y < z) \rightarrow x < z)$,
6 $\forall x \forall y\, (feuille\, x \wedge feuille\, y) \rightarrow (x < y \vee x = y \vee y < x)$,
7 $\forall x \forall y\, (feuille\, x \wedge feuille\, y) \rightarrow (x < y \rightarrow (\exists z\, feuille\, z \wedge x < z \wedge z < y))$,
8 $\forall x\, feuille\, x \rightarrow (\exists y\, feuille\, y \wedge x < y)$,
9 $\forall x\, feuille\, x \rightarrow (\exists y\, feuille\, y \wedge y < x)$,
10 $\forall \bar{x}\, arbre\, f\bar{x}$,
11 $\forall x \forall y\, x < y \rightarrow (feuille\, x \wedge feuille\, y,)$
12 $\forall x\, feuille\, x \leftrightarrow \neg arbre\, x$,
13 $\exists x\, feuille\, x$,

où f, g sont des éléments distincts pris dans **F**, x, y, z des variables, $\bar{x}$ un vecteur composé de variables x_i toutes distinctes, $\bar{y}$ un vecteur composé de variables y_i toutes distinctes et $t_i(\bar{y}, \bar{x})$ un terme formé d'un élément de **F** suivi de variables prises dans $\bar{x}$ ou $\bar{y}$.

Les schémas d'axiomes 1, 2 et 3 reprennent les trois schémas d'axiomes de la théorie des arbres (Maher, 1988) : le schéma d'axiome 1, dit *d'explosion*, le schéma d'axiome 2, dit de *conflit de symboles* et le schéma d'axiome 3, dit de *solution unique*. Les axiomes 4, 5,..., 9 concernent la relation $<$, vue comme un *ordre strict* 4,5, *total* 6, *dense* 7 et *sans extrême* 8,9. Les axiomes 10, 11, 12 et 13 sont dits de *typage*.

4.2 Le modèle standard A de T_A

La théorie T_A a pour modèle A *les arbres aux feuilles ordonnées*, défini comme suit, à partir d'un ensemble **Q** disjoint de **F** et muni d'une relation d'ordre total dense sans extrêmes :

Signature de A La signature de A est la même que celle de T_A.

Domaine de A Le domaine $\mathcal{A}$ de A est constitué des arbres[1](éventuellement infinis) construits sur $\mathbf{F} \cup \mathbf{Q}$. Chaque élément d'arité n de **F** est considéré comme une étiquette d'arité n et chaque élément de **Q** est considéré comme une étiquette d'arité 0.

Relations de A Au symbole de relation unaire $feuille$ on associe l'ensemble $feuille^A$ des arbres qui n'ont qu'un seul nœud. Au symbole de relation unaire $arbre$ on associe l'ensemble $arbre^A$ des arbres qui ont plus d'un nœud. Au symbole de relation binaire $<$ on associe l'ensemble des couples (x, y) tels que $x \in feuille^A$, $y \in feuille^A$ et la valeur de x soit strictement inférieure à la valeur de y.

[1]Pour définir formellement un arbre a construit sur un ensemble E, d'éléments de diverses arités, on définit d'abord un *nœud* comme un mot construit sur l'ensemble des entiers positifs. Un tel arbre a est alors une application de type $N \rightarrow E$, où N est un ensemble non vide de nœuds dont chaque élément $i_1 \ldots i_k$ (avec $k \geq 0$) respecte deux conditions : (1) si $k > 0$ alors $i_1 \ldots i_{k-1} \in N$, (2) si l'arité de $a(i_1 \ldots i_k)$ est n alors, l'ensemble des nœuds de N de la forme $i_1 \ldots i_k i_{k+1}$ s'obtient en donnant à i_{k+1} les valeurs $1, \ldots, n$.

Opérations de $\boldsymbol{A}$ A chaque symbole $f \in \mathbf{F}$ d'arité n on associe l'opération de *construction*[2] $f^A : \mathcal{A}^n \to \mathcal{A}$, où $f(a_1, \ldots, a_n)$, est l'arbre dont la racine est étiquetée f et qui a pour suite de fils $a_1, \ldots, a_n$.

La seule difficulté pour montrer que A est un modèle de T_A, est de montrer que l'axiome 3 de solution unique de T_A est vrai dans A. Pour ce point on peut reprendre une démonstration donnée en (Hanh, 2000).

4.3 Briques de T_A

Définition 4.3.1
On appelle brique*, toute conjonction α de formules atomiques à plat, telle que toute variable x figurant dans α, ait au moins une occurrence dans une sous-formule de α de la forme $feuille\ x$ ou $arbre\ x$. Une brique α sans occurrence du symbole " $=$ " est dite* relationnelle*. Une brique α sans occurrence du symbole " $<$ " et où chaque variable a une occurrence dans au moins une des équations de α est dite* équationnelle*.*

Définition 4.3.2
Si la brique α a une sous-formule de la forme

$$x_0 = t_0(x_1) \wedge x_1 = t_1(x_2) \wedge \ldots \wedge x_{n-1} = t_{n-1}(x_n)$$

avec x_{i+1} figurant dans le terme $t_i(x_{i+1})$, alors la variable x_n ainsi que l'équation $x_{n-1} = t_{n-1}(x_n)$ sont dites accessibles *dans α à partir de x_0.*

Du fait des schémas d'axiomes 1 et 2 on a la propriété suivante

Propriété 4.3.3
Soit α une brique. Si toutes les variables de $\bar{x}$ sont accessibles dans α à partir des variables libres de $\exists \bar{x} \alpha$ alors $T_A \models \exists ? \bar{x} \alpha$.

4.4 Briques résolues

On suppose que les variables de $\mathbf{V}$ sont ordonnées par une relation d'ordre total strict $\succ$.

Définition 4.4.1
On appelle brique bien typée *toute brique α telle que*

- *α ne contient pas de sous-formules de la forme $feuille\ x \wedge arbre\ x$,*
- *α ne contient pas de sous-formules de la forme $x = f\bar{y} \wedge feuille\ x$,*
- *α ne contient pas de sous-formules de la forme $x = y \wedge feuille\ x \wedge arbre\ y$ ou $x = y \wedge arbre\ x \wedge feuille\ y$,*
- *α ne contient pas de sous-formules de la forme $x < y \wedge arbre\ x$ ou $y < x \wedge arbre\ x$.*

[2] Formellement, l'opération *de construction* associée au symbole n-aire f de $\mathbf{F}$ est l'application $(a_1, \ldots, a_n) \mapsto a$, où les a_i sont des arbres quelconques et a est l'arbre défini comme suit à partir des a_i et de leurs ensembles E_i de nœuds : l'ensemble E de nœuds de a est $\{\varepsilon\} \cup \{ix \mid x \in E_i \text{ et } i \in 1..n\}$ et, pour tout $x \in E$, si $x = \varepsilon$, on a $a(x) = f$ et si x est de la forme iy, avec i un entier, $a(x) = a_i(y)$.

Définition 4.4.2
On appelle brique $(\succ)$-résolue dans T_A, toute brique α, telle que

1. *α est bien typée,*
2. *α ne contient pas de sous-formules de la forme $\beta \wedge faux$, où β est une formule différente de la formule $vrai$,*
3. *si $x = y$ est une sous-formule de α, alors $x \succ y$,*
4. *tous les membres gauches des équations de α sont distincts,*
5. *si $x < y$ est une sous-formule de α alors x et y n'apparaissent dans aucun membre gauche des équations de α,*
6. *α ne contient pas de sous-formules de la forme*

$$x_0 < x_1 \wedge x_1 < x_2 \wedge ... \wedge x_{n-1} < x_n \wedge x_n < x_0.$$

Propriété 4.4.3
Toute brique est équivalente dans T_A à une brique $(\succ)$-résolue.

Pour montrer ce théorème nous introduisons l'ensemble des règles de réécriture suivantes :

$$\begin{array}{lll}
(1) & y = f\bar{x} \wedge feuille\, y & \Longrightarrow faux, \\
(2) & x < y \wedge arbre\, x & \Longrightarrow faux, \\
(3) & y < x \wedge arbre\, x & \Longrightarrow faux, \\
(4) & x = y \wedge feuille\, x \wedge arbre\, y & \Longrightarrow faux, \\
(5) & x = y \wedge feuille\, y \wedge arbre\, x & \Longrightarrow faux, \\
(6) & feuille\, x \wedge arbre\, x & \Longrightarrow faux, \\
(7) & faux \wedge \alpha & \Longrightarrow faux, \\
(8) & x = fy_1...y_m \wedge x = gz_1...z_n & \Longrightarrow faux, \\
(9) & x = fy_1...y_n \wedge x = fz_1...z_n & \Longrightarrow x = fy_1...y_n \wedge \bigwedge_{i\in 1..n} y_i = z_i, \\
(10) & x_0 < x_1 \wedge ... \wedge x_{n-1} < x_n \wedge x_n < x_0 & \Longrightarrow faux, \\
(11) & x = x & \Longrightarrow vrai, \\
(12) & y = x & \Longrightarrow x = y, \\
(13) & x = y \wedge x = z & \Longrightarrow x = y \wedge y = z, \\
(14) & x = y \wedge x = fz_1...z_n & \Longrightarrow x = y \wedge y = fz_1...z_n, \\
(15) & x = y \wedge x < z & \Longrightarrow x = y \wedge y < z, \\
(16) & x = y \wedge z < x & \Longrightarrow x = y \wedge z < y,
\end{array}$$

où f et g sont des éléments distincts pris dans **F**, x, y, z des variables, $\bar{x}$ un vecteur de variables toutes distinctes et α une formule quelconque. Les règles (12),...,(16) sont conditionnées par $x \succ y$.

L'application d'une règle $\varphi \Longrightarrow \psi$ à la formule α consiste à remplacer dans α, une sous-formule φ par la formule ψ, et cela en considérant que le connecteur $\wedge$ soit associatif et commutatif. On a alors la propriété suivante :

Propriété 4.4.4
Toute application répétée des règles de réécriture précédentes, sur une brique, se termine et produit une brique $(\succ)$-résolue, équivalente dans T_A.

4.5 Complétude de T_A

Théorème 4.5.1
La théorie T_A est complète.

Nous montrons que T_A satisfait aux cinq conditions du théorème 3.3.1. Il suffit de choisir les ensembles $\Psi(u)$, A, A' et A'' de la manière suivante :

- $\Psi(u)$ est l'ensemble des formules de la forme $\exists \bar{y}\, u = f\bar{y}$, avec $f \in \mathbf{F}$,
- A est l'ensemble des briques,
- A' est l'ensemble des formules de la forme $\exists \bar{x}'\alpha'$, où
  - α' est une brique équationnelle $(\succ)$-résolue différente de la formule $faux$ où l'ordre $\succ$ est tel que toutes les variables de $\bar{x}'$ sont plus grandes que les variables libres de $\exists \bar{x}'\alpha'$,
  - toutes les équations de α' et toutes les variables de $\bar{x}'$ sont accessibles dans α' à partir des variables libres de $\exists \bar{x}'\alpha'$,
- A'' est l'ensemble des briques relationnelles $(\succ)$-résolues qui sont différentes de la formule $faux$.

Notons l'importance des briques $(\succ)$-résolues, qui sont à la base des ensembles A' et A''. Ce sont ces mêmes briques que l'on retrouvera dans l'algorithme général de résolution de contraintes quantifiées dans T_A.

5 Conclusion

Nous avons établi un théorème général qui donne une condition suffisante pour qu'une théorie du premier ordre soit complète et nous l'avons utilisé pour montrer que la théorie T_A des arbres aux feuilles ordonnées était complète. Nous avons conçu cette théorie pour des besoins théoriques et pratiques en représentation des connaissances et en résolution de contraintes quantifiées dans des structures hybrides à base d'arbres et de numériques ordonnés.

Dans l'immédiat nous travaillons sur une axiomatisation complète du modèle de Prolog III : l'algèbre des arbres avec des feuilles se comportant comme des rationnels additifs ordonnés (Colmerauer, 1990). Compte tenu du travail présenté ici et du résultat (Djelloul, 2003) sur le mélange d'arbres et de rationnels additifs, nous ne devrions pas rencontrer de difficultés. Nous développons également des algorithmes généraux de résolution de contraintes quantifiées dans différentes structures hybrides (y compris celle présentée dans cet article) tout en essayant de trouver des classes de complexité intéressantes. Ces algorithmes se déduisent facilement de la preuve de notre théorème de complétude 3.3.1. L'idée principale repose sur la manipulation des briques $(\succ)$-résolues et les propriétés des quantificateurs $\exists ?$, $\exists !$, $\exists_o^{\Psi(u)}{}_\infty$.

Remerciements Je remercie Alain Colmerauer pour nos nombreuses discussions et son aide dans l'organisation et la rédaction de ce papier. Je le remercie aussi pour les définitions et démonstrations de ses notes de cours de DEA.

Références

CHANG C. & KEISLER H. (1988). Model theory. section 1.4 theories and examples of theories. In ELSVIER, Ed., *Model theory*.

COLMERAUER A. (1984). Equations and inequations on finite and infinite trees. In *International conference on the fifth generation of computer systems Tokyo*, p. 85–99.

COLMERAUER A. (1990). An introduction to prolog 3. In *Communication of the ACM 33(7)*, p. 68–90.

COURCELLE B. (1983). Fundamental properties of infinite trees. In *Theretical Computer Science, vol. 25*.

COURCELLE B. (1986). Equivalences and transformations of regular systems applications to program schemes and grammars. In *Theretical Computer Science, vol. 42*.

DJELLOUL K. (2003). Intégration des nombres rationnels additifs et des arbres constructibles dans une théorie du 1er ordre complète. In *Mémoire DEA d'Informatique. Université de la Méditerrannée*, p. 62–83.

HANH D.-T.-B. (2000). Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis. In *Thèse d'informatique. Université de la méditerranée*.

HUET G. (1976). Resolution d'equations dans les langages d'ordre 1, 2,... ω. In *These d'Etat, Universite Paris 7. France*.

MAHER M. J. (1988). Complete axiomatization of the algebra of infinite, rational and infinite trees. In *Technical report, IBM - T.J.Watson Research Center*.

Deux opérateurs logiques adaptés à la représentation et à la manipulation des préférences

Gautier Meyer[1], Vincent Louis[1], Jean-Paul Sansonnet[2], Yannick Larvor[1]

[1] TECH/EASY, France Télécom R&D
2, avenue Pierre Marzin, 22307 Lannion, France
gautier.meyer@francetelecom.com
vincent.louis@francetelecom.com
yannick.larvor@francetelecom.com

[2] LIMSI-CNRS
Bât 508, BP 133, 91403 Orsay cedex, France
jean-paul.sansonnet@limsi.fr

Résumé : Les modèles d'agents rationnels cognitifs explicitent trop rarement l'étape de décision. Ce manque de formalisation est dû à l'absence d'outils, aussi bien sur le plan qualitatif que quantitatif, permettant d'évaluer et/ou comparer chacune des réactions possibles de l'agent. Ceci débouche souvent sur des mises en œuvre difficiles et/ou spécifiques. Cet article vise à combler ce vide en proposant une formalisation qualitative de la notion de *préférence*. En particulier, cet article définit deux opérateurs logiques pour d'une part, spécifier naturellement des préférences et d'autre part, comparer les réactions d'un agent deux à deux.

Mots-clés : Décision, préférence, modélisation, représentation des connaissances.

1 Introduction

Le paradigme agent propose un point de vue attrayant pour modéliser les systèmes informatiques complexes. Chaque entité du système informatique global est représentée de façon abstraite mais néanmoins avec des notions intuitives. En particulier, les agents rationnels peuvent être formalisés comme des entités cognitives à partir de notions primitives (appelées *attitudes mentales*) telles que la croyance, le désir ou l'intention (Rao & Georgeff, 1991; Sadek, 1992), et mis en œuvre par des mécanismes d'inférence automatique (Bretier & Sadek, 1997).

Un agent rationnel est avant tout une entité qui perçoit son environnement et qui agit sur celui-ci selon des principes de rationalité. En général, ces principes ne spécifient pas complètement les réactions d'un agent et en particulier dans les situations complexes où il a plusieurs motivations (ou intentions) et/ou plusieurs moyens de les satisfaire (plans d'action). Par exemple, dans le cadre d'une interaction multimodale, un agent

assistant qui fournit un itinéraire à un utilisateur peut choisir entre une modalité vocale (en énonçant les instructions) ou graphique (en montrant un plan) selon le contexte (cas de plusieurs plans satisfaisant la même intention).

C'est pourquoi les auteurs de (Sadek, 1991; Haddawy & Hanks, 1993) ont affirmé la nécessité de faire explicitement le lien entre l'état mental et la réaction de l'agent. Par la suite, les auteurs de (Haddawy & Hanks, 1993; Louis, 2002) ont montré que ce lien peut se diviser en deux phases distinctes et indépendantes (voir figure 1) : une phase de planification (Allen *et al.*, 1991; Louis, 2002) et une phase de décision (Doyle & Wellman, 1994; Chiclana *et al.*, 1996; Ha, 2001). La première vise à générer les réactions potentielles de l'agent (i.e. les alternatives) tandis que la seconde cherche à ordonner ces dernières pour choisir la réaction effective à mettre en œuvre.

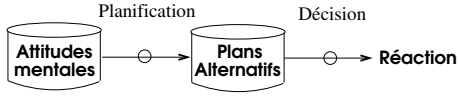


FIG. 1 – Lien général entre état mental et réactions

Comme le montre le travail de (Lang, 2004), la phase de décision a été envisagée de nombreuses façons et en particulier de façon quantitative et qualitative. Dans l'approche quantitative, une *fonction d'utilité* est généralement employée pour classer les alternatives. L'utilité espérée de chaque alternative est calculée et celle ayant la valeur la plus forte est choisie comme réaction. C'est une méthode très précise pour représenter les connaissances sur le classement des alternatives. De plus, elle permet de prendre en compte la nature stochastique de l'environnement (notion d'utilité espérée) et est développée depuis longtemps dans le contexte de la théorie de la décision. Malheureusement les quatre inconvénients suivants sont souvent reprochés à cette méthode :

– La détermination de l'utilité est en pratique difficile. Elle nécessite un effort de spécification très important puisqu'elle demande de décrire précisément chaque alternative en ayant à l'esprit le problème dans sa globalité (Doyle & Wellman, 1994; Ha & Haddawy, 1999; Faltings *et al.*, 2004).
– Si « l'optimalité » du résultat n'est pas cruciale, cette méthode ne se justifie pas particulièrement (Brafman & Tennenholtz, 1997; Boutilier *et al.*, 2004; Ha & Haddawy, 1999).
– Afin d'avoir une méthode de décision plus homogène avec la description interne de l'agent via des attitudes mentales, une technique *qualitative* plutôt que *quantitative* serait préférable (Dubois *et al.*, 2001).
– Enfin, si l'on s'inspire du modèle humain pour modéliser les agents, il semble plus naturel de faire appel à des techniques de décision qualitatives (Brafman & Tennenholtz, 1997; Rossi *et al.*, 2004).

Afin de contourner ces quatre limitations, nous choisissons de baser le classement des alternatives sur des informations qualitatives : les préférences. Plus précisément nous définissons formellement une notion appelée *préférence partielle* qui permet de comparer, selon un « point de vue », les alternatives deux à deux sans recourir à des représentations numériques. Cette notion, a priori simple et intuitive (voir la section

suivante), permet de spécifier facilement les informations élémentaires nécessaires au classement et donc à la phase de décision.

L'article suit l'organisation suivante. La section 2 décrit de façon globale notre proposition. Les sections 3 et 4 formalisent la construction progressive de notre modèle de préférence en introduisant deux opérateurs logiques : un opérateur relatif à la préférence partielle primitive et un autre relatif à la préférence partielle étendue.

2 Schéma général

L'objectif de cet article est de proposer un formalisme permettant, d'une part, de spécifier des préférences de manière concise et naturelle et d'autre part de comparer des alternatives deux à deux. Pour ce faire, les informations nécessaires au choix sont modélisées grâce à un nouveau concept : la préférence partielle.

Nombreux sont les chercheurs en IA et en philosophie à penser que la spécification intuitive d'une préférence fait implicitement l'hypothèse *Ceteris Paribus* (voir (Boutilier *et al.*, 2004)) : tous les aspects autres que ceux explicités dans la préférence spécifiée sont considérés comme étant égaux par ailleurs (Hansson, 1996; Doyle & Wellman, 1994). Par exemple, le fait d'exprimer que les plats à base de poissons sont préférés aux plats à base de viandes signifie implicitement que les caractéristiques comme le prix et l'abondance sont les mêmes pour les deux plats comparés. De plus, Coste-Marquis *et al.* (2004) ont montré que la spécification de préférences à l'aide de hypothèse *Ceteris Paribus* permet d'exprimer de façon succincte de nombreux pré-ordres partiels. Cependant une telle hypothèse ne permet pas de comparer tous les objets entre eux et en particulier ceux dont les caractéristiques (non spécifiées par la préférence) sont différentes : un poisson à la vapeur est-il préférable à une viande grillée ?

Pour contourner ce problème, l'idée directrice de notre approche consiste à scinder la phase de spécification des préférences en deux étapes (voir la figure 2). Dans un premier

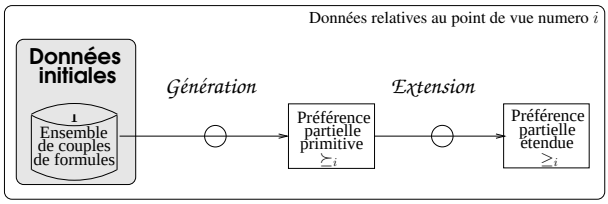


FIG. 2 – Construction des préférences.

temps, les données initiales définissant chaque préférence partielle sont interprétées selon l'hypothèse *Ceteris Paribus*, ce qui génère une *préférence partielle primitive* (phase dite de génération). Dans un deuxième temps, chaque préférence partielle primitive est étendue (au-delà de l'hypothèse *Ceteris Paribus*) en une *préférence partielle étendue*. Cette phase dite « d'extension » permet de comparer n'importe quel couple d'objets. Il est à remarquer que plusieurs préférences partielles peuvent être spécifiées en parallèle. Chacune d'elles représente un « point de vue » pour classer les alternatives.

Dans notre approche, chaque préférence est modélisée par une relation binaire entre les objets à comparer. Dans le contexte des agents rationnels auquel nous nous intéressons, ces objets correspondent à des états du monde, que nous appellerons « alternatives ». Or il est difficile voire impossible de représenter exhaustivement toutes les caractéristiques d'un état du monde (voir par exemple le problème du cadre (McCarthy & Hayes, 1969)). Notre modèle associe donc en outre à chaque préférence un opérateur logique portant sur des couples de formules décrivant partiellement les états à comparer. En résumé, chaque préférence partielle est associée à la fois à une relation binaire sur l'ensemble des alternatives et à un opérateur sur des couples de formules logiques représentant les alternatives. Dans ce sens, notre vision est proche de celle de Von Wright selon laquelle les préférences sont spécifiées par des descriptions mais s'appliquent à des alternatives « concrètes » correspondant à ces descriptions (von Wright, 1972).

3 Préliminaires

3.1 Langage utilisé

Dans cet article, $\mathscr{L}$ est un langage propositionnel construit sur un ensemble de symboles propositionnels SYM, sur les constantes propositionnelle $\top$ (tautologie) et $\bot$ (contradiction), et au moyen des opérateurs logiques classiques ($\wedge, \vee, \neg, \Leftrightarrow, \Rightarrow$). Les symboles $\phi, \phi_1, \ldots \psi, \psi_1 \ldots$ dénotent des formules quelconques de $\mathscr{L}$. La notation $\models \phi$ signifie que la formule ϕ est valide. Pour toute formule $\phi \in \mathscr{L}$, $\mathrm{S}(\phi)$ dénote l'ensemble des symboles propositionnels autres que $\top$ et $\bot$ qui apparaissent dans cette formule. Il est à remarquer que $\top$ et $\bot$ sont des formules de $\mathscr{L}$ et que les ensembles $\mathrm{S}(\top)$ et $\mathrm{S}(\bot)$ sont vides par définition. $\mathrm{S}_m(\phi)$ dénote l'ensemble des symboles propositionnels autres que $\top$ et $\bot$ qui apparaissent dans la formule ϕ et qui sont pertinents pour la valeur de vérité de cette formule. Plus précisément, il représente l'ensemble[1] des symboles propositionnels qui apparaissent dans la plus « concise » des formules logiquement équivalentes à ϕ : pour toutes formules ϕ et ψ de $\mathscr{L}$,

$$\mathrm{S}_m(\phi) = \mathrm{S}(\psi) \quad \text{ssi} \quad \begin{array}{ll} \bullet & \models \psi \Leftrightarrow \phi \\ \bullet & \forall \psi' \in \mathscr{L}, \\ & \quad \text{si} \quad \models \psi' \Leftrightarrow \psi \\ & \quad \text{alors} \quad \mathrm{S}(\psi) \subseteq \mathrm{S}(\psi') \end{array}$$

3.2 Alternatives

Une alternative est un *état du monde*, i.e. une interprétation particulière de tous les symboles propositionnels de SYM. Nous les notons $a, a_1, \ldots, b, \ldots$ ALT dénote l'ensemble des alternatives. Formellement, pour toute alternative a de ALT, et pour tout symbole s de SYM, soit $a \models s$ soit $a \models \neg s$.

Une formule ϕ de $\mathscr{L}$ désigne en général un ensemble de combinaisons de valeurs de vérité pour les symboles propositionnels de SYM et donc un ensemble d'alterna-

[1] On peut montrer par l'absurde que deux formules logiquement équivalentes ont le même ensemble S_m et que donc celui-ci est unique pour une formule donnée.

tives : $\{a$ tel que $a \in$ ALT et $\models \phi\}$. En particulier la formule $\top$ désigne l'ensemble de toutes les alternatives et $\bot$ ne désigne aucune alternative. De plus, lorsqu'une formule du langage $\mathscr{L}$ spécifie une valeur de vérité particulière pour chacun des symboles propositionnels de SYM, cette formule représente une unique alternative. Nous notons f_a la formule canonique qui représente uniquement l'alternative a.

Intuitivement, chaque alternative désigne un objet qui peut être comparé par les préférences. Le terme *alternative* est employé ici pour bien mettre en évidence la notion de choix qui y est associée.

3.3 Préférence

Une *préférence* est modélisée par une relation binaire définie sur l'ensemble des alternatives. En particulier, une *préférence partielle* est une relation de préférence circonscrite à un point de vue. Nous noterons par la suite de telles relations avec les symboles $G_1, G_2, \ldots, G_i, G'_1, \ldots$ où i désigne le numéro du point de vue associé. De façon générale, la notation $a_1 G_i a_2$ signifie que, du point de vue numéro i, l'alternative a_1 est préférée à l'alternative a_2.

Une *préférence* peut donc être vue comme un ensemble de couples d'alternatives. Cet ensemble peut contenir des couples contradictoires entre eux, par exemple, à la fois aG_ib et bG_ia.

3.4 Indifférence

Deux alternatives sont dites *indifférentes* suivant une préférence partielle donnée (et donc selon un point de vue) si et seulement si chacune d'elles est préférée à l'autre suivant la préférence partielle considérée ou bien si aucune d'elles n'est préférée à l'autre suivant la préférence partielle considérée.

4 Préférences partielles primitives

4.1 Formalisation

4.1.1 Ceteris Paribus :

Deux alternatives a et b sont dites *Ceteris Paribus* vis-à-vis des deux formules ϕ et ψ de $\mathscr{L}$ (noté $CP(a, b, \phi, \psi)$), si et seulement si les alternatives a et b attribuent la même valeur à tout symbole qui n'apparaît ni dans ϕ ni dans ψ. Formellement :

$$CP(a,b,\phi,\psi) \text{ ssi } \begin{bmatrix} \forall s \in \text{SYM}, \\ \text{si} \quad s \notin \text{S}_m(\phi) \text{ et } s \notin \text{S}_m(\psi) \\ \text{alors} \quad a \models s \text{ ssi } b \models s \end{bmatrix} \tag{1}$$

4.1.2 Définition :

A chaque relation binaire de *préférence partielle primitive* G_i (telle que définie dans la section 3.3) est associé un opérateur logique défini sur l'ensemble des formules de

$\mathscr{L}$. Nous le noterons avec le symbole $\succeq_i$ lorsque la relation correspondante est associée au point de vue i. L'ensemble des opérateurs $\succeq_i$ définissent le langage $\mathscr{L}'$: si ϕ et ψ sont des formules de $\mathscr{L}$ (et non $\mathscr{L}'$) alors la formule $\phi \succeq_i \psi$ est une formule de $\mathscr{L}'$. $\phi \succeq_i \psi$ signifie que les alternatives qui vérifient la formule ϕ sont préférées, suivant le point de vue numéro i, aux alternatives qui vérifient la formule ψ si elles sont Ceteris Paribus (identiques par ailleurs). La sémantique de cet opérateur est formalisée comme suit :

$$\models \phi \succeq_i \psi \quad \text{ssi} \quad \bullet \quad \exists (a,b), \text{ tel que } \left| \begin{array}{l} a \models \phi \\ b \models \psi \end{array} \right.$$

$$\bullet \quad \left[\begin{array}{l} \forall (a,b), \\ \text{si} \left| \begin{array}{l} a \models \phi \\ b \models \psi \\ CP(a,b,\phi,\psi) \end{array} \right. \\ \text{alors} \quad aG_ib \end{array} \right] \tag{2}$$

Dans la pratique, chaque préférence partielle primitive G_i est générée à partir d'un ensemble initial de formules de la forme $\phi \succeq_i \psi$, où ϕ et ψ sont des formules de $\mathscr{L}$: ce sont les données initiales représentées dans la figure 2. Ces formules n'induisent pas forcément des ensembles de préférences cohérents : elles peuvent spécifier à la fois que $\phi \succeq_i \psi$ et que $\psi \succeq_i \phi$.

Afin que la relation binaire (sur les alternatives) support de chaque *préférence partielle primitive*, définie comme la sémantique de l'opérateur de *préférence partielle primitive* (sur les formules de $\mathscr{L}$), permette de comparer naturellement les alternatives, elle doit être un pré-ordre[2] (i.e. une relation réflexive et transitive). C'est pourquoi les propriétés supplémentaires suivantes sont imposées à chaque opérateur de préférence partielle primitive.

4.1.3 Défaut (D) :

Par défaut, si ce n'est pas contradictoire avec le reste de la préférence partielle primitive, une formule ϕ de $\mathscr{L}$ n'est pas préférée à une formule ψ de $\mathscr{L}$. Formellement le défaut normal sans pré-requis (à la Reiter (1980)) suivant est imposé :

$$\top : \neg(\phi \succeq_i \psi) \rightarrow \neg(\phi \succeq_i \psi) \tag{3}$$

4.1.4 Réflexive (R) :

On impose que toute formule de $\mathscr{L}$ non inconsistante est préférée à elle même :

$$\text{si } \not\models \phi \Leftrightarrow \bot \quad \text{alors} \quad \models \phi \succeq_i \phi \tag{4}$$

Ceci entraîne que chaque relation binaire de *préférence partielle primitive* associée G_i est réflexive. Cela permet donc d'exprimer le fait que deux alternatives semblables sont indifférentes.

[2]Cette relation n'est pas nécessairement un ordre car deux alternatives indifférentes ne sont pas nécessairement égales.

4.1.5 Transitive (T) :

On impose que si un agent préfère une formule ϕ_1 à une formule ϕ_2 ainsi que la formule ϕ_2 à une formule ϕ_3, alors il préfère la formule ϕ_1 à la formule ϕ_3. C'est ce que formalise la propriété suivante :

$$\text{si } \models \phi_1 \succeq_i \phi_2 \text{ et } \models \phi_2 \succeq_i \phi_3 \quad \text{alors } \models \phi_1 \succeq_i \phi_3 \tag{5}$$

Ceci entraîne que la relation binaire de *préférence partielle primitive* associée G_i est transitive. Cela permet donc d'exprimer le fait que si un agent préfère une alternative a_1 à une alternative a_2 ainsi que l'alternative a_2 à une alternative a_3 alors il préfère l'alternative a_1 à l'alternative a_3.

4.2 Propriétés remarquables

4.2.1 Lien entre l'opérateur $\succeq_i$ et la relation G_i

Comme énoncé dans la section 3.2, à chaque alternative a de ALT est associée une formule de $\mathscr{L}$ (notée f_a) spécifiant une valeur pour chacun des symboles de $\mathscr{L}$ et telle que seule l'alternative a valide cette formule. Par conséquent, si un couple d'alternatives appartient à une relation de préférence alors les formules associées sont reliées par l'opérateur logique correspondant. Formellement, pour tout couple d'alternatives (a, b) :

$$\text{si} \quad aG_ib \quad \text{alors} \quad \models f_a \succeq_i f_b \tag{6}$$

Preuve immédiate en identifiant dans la définition (2) la formule ϕ avec la formule f_a et la formule ϕ avec la formule f_b.

4.2.2 Comportement des opérateurs logique $\succeq_i$ avec l'équivalence logique :

Si deux formules logiques sont équivalentes alors toute occurrence de l'une dans une préférence partielle peut-être substituée par une occurrence de l'autre et inversement. Formellement,

$$\text{si} \left| \begin{array}{l} \models \phi_1 \Leftrightarrow \phi_2 \\ \models \psi_1 \Leftrightarrow \psi_2 \\ \models \phi_1 \succeq_i \psi_1 \end{array} \right. \quad \text{alors} \quad \models \phi_2 \succeq_i \psi_2 \tag{7}$$

Par conséquent, si deux formules sont équivalentes alors la première est préférée à la seconde et inversement. Preuve en utilisant les propriétés de réflexivité et de transitivité de l'opérateur logique.

4.2.3 *Ceteris paribus* (CP) :

Une formule ϕ est préférée à une formule ψ dès lors qu'une caractéristique de ϕ est préférée à une caractéristique de ψ et que ϕ et ψ sont égales par ailleurs. Par conséquent, si deux formules sont liées par une relation de préférence partielle primitive alors la

spécialisation de l'une par n'importe quel critère non encore spécifié et la spécialisation de l'autre suivant le même critère sont liées par cette relation. Formellement :

$$\text{si } \models \phi \succeq_i \psi \quad \text{alors} \quad \left[\begin{array}{l} \forall \gamma \in \mathscr{L} \\ \text{si} \left| \begin{array}{l} \gamma \wedge \phi \not\models \bot \\ \gamma \wedge \psi \not\models \bot \end{array} \right. \\ \text{alors} \quad \models (\phi \wedge \gamma) \succeq_i (\psi \wedge \gamma) \end{array}\right] \tag{8}$$

Preuve : en utilisant la définition 2.

4.2.4 Addition des préférences :

Si les alternatives vérifiant la formule ϕ sont préférées à celles vérifiant la formule ψ lorsqu'elles sont par ailleurs égales, et si, de la même façon, les alternatives vérifiant la formule γ sont préférées à celles vérifiant la formule ω lorsqu'elles sont par ailleurs égales, alors les alternatives vérifiant à la fois les formules ϕ et γ sont préférées aux alternatives vérifiant à la fois les formules ψ et ω lorsqu'elles sont par ailleurs égales :

$$\text{si } \models \phi \succeq_i \psi \text{ et } \models \gamma \succeq_i \omega \quad \text{alors } \models (\phi \wedge \gamma) \succeq_i (\psi \wedge \omega) \tag{9}$$

Preuve : en utilisant les propriétés Ceteris Paribus et de transitivité.

4.3 Exemple

Soit le langage propositionnel $\mathscr{L}$ construit sur l'ensemble des 5 symboles propositionnels SYM$= \{C_v, C_b, C_r, C_o, C_j\}$ (signifiant respectivement « vert » , « bleu » , « rouge » , « orange » , « jaune »), comme indiqué section 3 . Ce langage permet de décrire quelques propriétés du monde (i.e. les couleurs). Considérons le cas où un point de vue (i.e. critère) « couleur » a été défini grâce aux informations regroupées dans le tableau de la figure 3. La notation « $C_v \succeq_1 C_b$ » signifie que tout objet de couleur verte est préféré à tout autre objet de couleur bleue s'ils sont égaux par ailleurs (hypothèse Ceteris Paribus). Ces informations, du fait des propriétés de l'opérateur de préférence

couleur		
C_v	$\succeq_1$	C_b
C_r	$\succeq_1$	C_v
C_o	$\succeq_1$	C_j
C_j	$\succeq_1$	C_v
C_o	$\succeq_1$	C_v

FIG. 3 – Informations spécifiées.

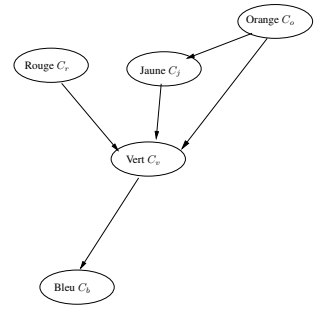


FIG. 4 – Représentation graphique.

partielle primitive $\succeq_i$, impliquent que :

– $C_r \succeq_1 C_r$, $C_b \succeq_1 C_b$, ... (Réflexivité)
– $C_r \succeq_1 C_b$, $C_o \succeq_1 C_b$, $C_j \succeq_1 C_b$ (Transitivité)
– $C_v \wedge C_r \succeq_1 C_b \wedge C_r$, $C_v \wedge C_o \succeq_1 C_b \wedge C_o$, ... (Ceteris Paribus)

Par exemple, le fait que, du point de vue « couleur », les objets rouges sont préférés aux bleus (i.e. $C_r \succeq_1 C_b$) peut être explicitement déduit des deux premières lignes du tableau et de la propriété de transitivité de l'opérateur de préférence partielle primitive. Les données du tableau peuvent aussi être représentées de façon graphique (voir figure 4) afin d'être plus lisibles.

5 Préférences partielles étendues

5.1 Motivation

Les propriétés que nous avons énoncées dans les sections précédentes permettent de construire pratiquement des opérateurs de préférence partielle primitive (sur des couples de formules) et donc de spécifier commodément une relation binaire de préférence partielle (sur des alternatives).

Néanmoins, dans la pratique, il est fréquent de comparer des alternatives qui ne sont pas explicitement en relation via une préférence partielle primitive. En effet, les critères sont généralement interdépendants et donc une différence entre alternatives sur un critère implique des différences sur les autres critères[3]. Par conséquent, il est nécessaire de pouvoir comparer des couples d'alternatives tels que la première alternative vérifie la formule $\phi \wedge \gamma$ et la seconde la formule $\psi \wedge \omega$ et tels que seule la formule $(\phi \succeq_i \psi)$ de $\mathscr{L}'$ a été spécifiée d'une manière ou d'une autre lors de la définition des préférences. En particulier, il faut un opérateur logique sur les formules qui puisse traiter les couples de formules de la forme $(\phi \wedge \gamma, \psi \wedge \omega)$ en sachant uniquement que $\phi \succeq_i \psi$ mais sans rien savoir au sujet du couple de formules (γ, ω).

5.2 Exemple (suite)

Considérons la situation issue de l'exemple précédent et telle que les termes V et M (signifiant respectivement « voiture » et « moto ») fassent aussi partie des symboles du langage $\mathscr{L}$. L'outil logique que nous venons de définir pour manipuler les préférences partielles primitives nous permet de déduire d'une telle situation que, par exemple, du point de vue de « couleur », les voitures vertes sont préférées aux voitures bleues ($C_r \wedge V \succeq_1 C_b \wedge V$) en se basant sur la formule $C_r \succeq_1 C_b$ précédemment inférée et sur la propriété CP. En d'autres termes, le fait qu'une alternative vérifiant le critère « rouge » est préférée à une alternative vérifiant le critère « bleu » si ces deux alternatives assignent aux autres critères les mêmes valeurs de vérité (Ceteris Paribus = toute chose égale par ailleurs) peut être explicitement déduit.

Malheureusement, si les autres critères n'ont pas les mêmes valeurs, une telle déduction ne peut avoir lieu. En particulier, rien ne peut être dit au sujet des préférences entre

[3]Par exemple, le point exact de départ d'un voyage peut dépendre du moyen de transport utilisé : un voyage depuis Paris peut avoir comme point de départ l'aéroport de Roissy ou la gare Montparnasse.

les alternatives vérifiant les critères « bleu » et « moto » et celles vérifiant les critères « rouge » et « voiture ».

5.3 Principe d'extension

Néanmoins, il est souvent nécessaire de pouvoir comparer n'importe quelle alternative sur la base des données initiales. C'est pourquoi nous faisons une nouvelle hypothèse pour étendre chaque *préférence partielle primitive* en une *préférence partielle étendue*. Cette hypothèse exprime le fait qu'une préférence est un « argument » pour préférer les alternatives vérifiant la formule ϕ aux alternatives vérifiant la formule ψ.

5.4 Formalisation

5.4.1 Définition :

A chaque relation binaire de *préférence partielle étendue* G'_i (telle que définie dans la section 3.3) est affecté un opérateur logique défini sur l'ensemble des formules de $\mathscr{L}$. Nous noterons un tel opérateur avec le symbole $\geq_i$ lorsque la relation correspondante est associée au point de vue i. De façon générale, la notation $\phi \geq_i \phi$ signifie que, du point de vue de la préférence numéro i, les alternatives vérifiant la formule ϕ sont préférées à celles vérifiant la formule ϕ (sans la restriction Ceteris Paribus). Formellement, la sémantique de cet opérateur est la suivante :

$$\models \phi \geq_i \psi \quad \text{ssi} \quad \bullet \quad \exists(a,b), \text{ tel que } \left| \begin{array}{l} a \models \phi \\ b \models \psi \end{array} \right.$$

$$\bullet \quad \left[\begin{array}{ll} \forall (a,b), & \\ \text{si} & \left| \begin{array}{l} a \models \phi \\ b \models \psi \end{array} \right. \\ \text{alors} & aG'_i b \end{array} \right] \tag{10}$$

5.4.2 Ceteris Imparibus (CI) :

La formule ϕ est préférée *Ceteris Imparibus* à la formule ψ selon le point de vue i (noté $\text{CI}_i(\phi, \psi)$), si et seulement si il existe un couple de formules (ϕ', ψ') tel que ϕ' est préférée à ψ' selon la préférence partielle primitive numéro i (noté $\phi' \succeq_i \psi'$) mais pas inversement, et que la formule ϕ' est subsumée par la formule ϕ et que la formule ψ' est subsumée par la formule ψ. Formellement :

$$\text{CI}_i(\phi, \psi) \quad \text{ssi} \quad \left[\begin{array}{ll} \exists(\phi', \psi') \in \mathscr{L} \times \mathscr{L} & \\ \text{tel que} & \left| \begin{array}{l} \models \phi' \succeq_i \psi' \\ \models \neg(\psi' \succeq_i \phi') \\ \models \phi \Rightarrow \phi' \\ \models \psi \Rightarrow \psi' \end{array} \right. \end{array} \right] \tag{11}$$

5.4.3 Règle de construction :

Chaque opérateur de *préférence partielle étendue* est construit comme l'extension *Ceteris Imparibus* d'une *préférence partielle primitive* :

$$\models \phi \succeq_i \psi \quad \text{ssi} \quad \left[\models \phi \succeq_i \psi \quad \text{ou} \quad \text{CI}_i(\phi, \psi) \right] \tag{12}$$

Les propriétés CI et T ne doivent pas être imposées simultanément aux opérateurs de préférence partielle étendue. En effet, si elles l'étaient, n'importe quelle alternative vérifiant une formule ϕ serait préférée (pour la relation de préférence associée à cet opérateur) à une autre alternative vérifiant ψ dès lors qu'il existerait deux formules ϕ_1 et ψ_1 telles que $\models \phi \succeq_i \phi_1$ et $\models \psi_1 \succeq_i \psi$. Preuve immédiate en utilisant les propriétés CI et T.

La condition $\neg(\psi \succeq_i \phi)$, spécifiée dans la propriété *Ceteris Imparibus*, permet de « bloquer » la règle de construction lorsque les formules ϕ et ψ sont tout aussi préférables. Sans cette condition, une telle règle entraînerait la symétrie de l'opérateur $\succeq_i$ et donc de la relation binaire G_i'. Preuve par l'absurde en utilisant les propriétés R et T.

Enfin, il est à remarquer que les relations binaires induites par les opérateurs $\succ_i$ et $\succeq_i$ sont différentes. Alors que la première (G_i) est transitive et réflexive, la seconde (G_i') n'est que réflexive. Cela peut se justifier par le fait que G_i permet d'interpréter les données d'entrée et que G_i' permet d'exploiter ces informations pour la décision.

5.4.4 Indifférence :

La propriété *Ceteris Imparibus* implique que deux alternatives sont indifférentes si elles ont chacune au moins une composante préférée :

$$\text{si} \left| \begin{array}{l} \models \phi \succeq_i \psi \\ \models \gamma \succeq_i \omega \end{array} \right. \quad \text{alors} \left| \begin{array}{l} \models (\phi \wedge \omega) \succeq_i (\psi \wedge \gamma) \\ \models (\psi \wedge \gamma) \succeq_i (\phi \wedge \omega) \end{array} \right. \tag{13}$$

5.5 Exemple (Fin)

Ce nouvel opérateur préserve toute les informations déjà déduites. Par exemple, les voitures rouges sont toujours préférées aux voitures bleues (i.e. la formule $C_r \wedge V \succeq_1 C_b \wedge V$ est vérifiée). De plus, il permet de déduire de nouvelles informations, plus intéressantes, au sujet de préférence sur des alternatives dont les caractéristiques qui les différencient n'apparaissent pas dans les données initiales. Par exemple, du point de vue « couleur », les voitures rouges sont préférées aux motos bleues (i.e. $C_r \wedge V \succeq_1 C_b \wedge M$). Cela signifie que, du point de vue « couleur », toute alternative vérifiant les propriétés « rouge » et « voiture » est préférée aux alternatives vérifiant les propriétés « bleu » et « moto ».

Le premier opérateur ($\succeq_1$) a permis de déduire les conséquences « logiques » des informations initialement spécifiées sur les couleurs (voir le tableau 3). Le second opérateur, quant à lui, permet d'utiliser ces déductions pour comparer des alternatives qui peuvent différer l'une de l'autre par des critères autres que la couleur. Ce dernier opérateur fournit donc un classement des alternatives en se focalisant uniquement sur le point de vue « couleur ».

6 Conclusion

La notion de préférence a été formalisée dans la littérature afin de dépasser une des limites de la théorie de la décision classique : la difficulté de déterminer la fonction d'utilité d'un système. Généralement employée sous l'hypothèse *ceteris paribus*, elle permet de représenter facilement des informations utiles pour classer les alternatives. Même si les informations ainsi décrites sont moins précises que si elles étaient décrites quantitativement, elles sont plus aisément récupérables[4] (elicitation phase).

Plus récemment, la notion de préférence a été traitée de manière qualitative. Dans (Boutilier *et al.*, 2004; Rossi *et al.*, 2004), la notion de changement élémentaire (*Flip*) permet de comparer deux alternatives. Ces travaux se basent sur la définition suivante : une alternative est préférée à une autre s'il existe une suite de changements élémentaires aggravante entre les deux. En d'autres termes a est préférée à b s'il existe une suite d'alternatives $\{a_i\}$ telle que $a_1 = a$, $a_n = b$ et telle que, quel que soit i, l'alternative a_i est obtenue de l'alternative a_{i-1} via un unique changement élémentaire et a_{i-1} est préférée à a_i. Malheureusement, l'existence d'une suite de changements élémentaires aggravante n'implique pas l'absence de suite de changements élémentaires améliorante. Que faire dans de tels cas ? De plus, comme les comparaisons entre les alternatives à départager se font via toutes les alternatives imaginables qui composent la suite aggravante, il ne nous semble pas que cette approche soit adaptée pour traiter les situations réelles. En effet, dans de telles situations les caractéristiques permettant de décrire chaque alternative sont potentiellement très nombreuses et donc la suite d'alternatives à considérer aussi.

Dans cet article nous avons traité le problème de gestion des préférences sous un angle nouveau en partageant la phase de spécification en deux. Chaque *préférence partielle primitive* est interprétée selon l'hypothèse *Ceteris Paribus*, ce qui permet de spécifier intuitivement ces préférences. Chacune d'elles est alors étendue par l'hypothèse *Ceteris Imparibus* en une *préférence partielle étendue* qui permet de comparer deux alternatives quelconques (non nécessairement égales par ailleurs) sans avoir à définir explicitement de notion d'équivalence entre leurs caractéristiques.

Un prolongement naturel de ce travail est d'utiliser cette méthode de spécification des préférences pour définir un ensemble de points de vue, puis de résoudre à chaque instant les contradictions éventuelles entre ces différents points de vue (i.e. préférences partielles) via une phase d'agrégation (définie sur le modèle de (Rossi *et al.*, 2004) par exemple). Cette extension est motivée par l'idée développée dans (Dubois *et al.*, 2001) selon laquelle la décision multi-critères et la décision multi-agents sont deux approches différentes de la même problématique. Chacune de nos *préférences partielles* jouera ainsi le rôle de la préférence d'un agent dans le modèle proposé par Rossi et al. Ce principe proposé dans (Meyer *et al.*, 2005), pourra être poursuivis pour, par exemple, étudier plus en détail les stratégies d'agrégation des points de vue pour la construction d'une préférence « globale ». Un autre axe d'évolution concerne la prise en compte d'informations hiérarchiques pour spécifier les préférences partielles. Par exemple, comment exprimer que la préférence « un monospace vert est préféré à un monospace rouge »

[4] Il est souvent naturel de spécifier la notion de « désirabilité » avec des informations qualitatives portant sur des alternatives types (ex : je préfère les repas avec du vin que ceux avec de l'eau).

a priorité sur la préférence « une voiture rouge est préférée à une voiture verte » ? Ce problème rejoint par certains aspects celui des défauts et des informations incomplètes dans le domaine de la représentation des connaissances. Enfin, nous envisageons d'utiliser le formalisme proposé dans cet article pour modéliser un nouveau type d'attitude mentale (en plus des croyances, désirs et intentions) des agents rationnels fondé sur la préférence. Cette nouvelle attitude mentale devrait permettre de formaliser explicitement la phase de choix des réactions de cet agent. En outre, il serait intéressant d'étudier également les liens qu'entretiennent préférence et désir, et notamment de vérifier si le désir d'une chose peut être appréhendé comme la préférence de cette chose par rapport à son contraire.

Références

ALLEN J. F., KAUTZ H. A., PELAVIN R. N. & TENENBERG J. D. (1991). *Reasoning about plan*. Morgan Kaufmann.

BOUTILIER C., BRAFMAN R. I., DOMSHLAK C., POOLE D. & HOOS H. H. (2004). Preference-based constraint optimization with CP-nets. *Computational Intelligence*, **20**(2), 137–157.

BRAFMAN R. I. & TENNENHOLTZ M. (1997). Modeling agents as qualitative decision makers. *Artificial Intelligence*, **94**.

BRETIER P. & SADEK D. (1997). A rational agent as the Kernel of a cooperative spoken dialogue system : Implementing a logical theory of interaction. In *Proceedings of the ECAI'96 Workshop on Agent Theories, Architectures, and Languages*, volume 1193 of *LNAI*, p. 189–204, Berlin : Springer.

CHICLANA F., HERRERA-VIEDMA E. & HERRERA F. (1996). Preference relations as the information representation base in multi-person decision making. In *IPMU'96*, volume 1, p. 459–464, Granada, Spain.

COSTE-MARQUIS S., LANG J., LIBERATORE P. & MARQUIS P. (2004). Expressive power and succinctness of propositional languages for preference representation. In *KR'04*, p. 203–212, Whistler, Canada.

DOYLE J. & WELLMAN M. P. (1994). Representing preferences as ceteris paribus comparatives. In *AAAI Symposium on Decision-Theoretic Planning*, Stanford, CA, USA : AAAI press.

DUBOIS D., FARGIER H., PERNY P. & PRADE H. (2001). Towards a qualitative multicriteria decision theory. In *Eurofuse Workshop on Preference Modeling and Applications*, p. 121–129, Granada, Spain.

FALTINGS B., TORRENS M. & PU P. (2004). Solution generation with qualitative models of preferences. *Computational Intelligence*, **20**(2), 246–263.

HA V. (2001). *Reasoning with Partial Preference Models*. PhD thesis, University of Wisconsin-Milwaukee, USA.

HA V. & HADDAWY P. (1999). A hybrid approach to reasoning with partial preference models. In *UAI'99*, p. 263–270, Stockholm, Sweden.

HADDAWY P. & HANKS S. (1993). *Utility Models for Goal-Directed Decision Theoretic Planners*. Rapport interne TR-93-06-04, Univ. of Washington, Dept. of Computer Science and Engineering, USA.

HANSSON S. O. (1996). What is ceteris paribus préférence. *Journal of Philosophical Logic*, **25**(3), 307–332.

LANG J. (2004). Logical preference representation and combinatorial vote. *Annals of Mathematics and Artificial Intelligence*, **42**(1-3), 37–71.

LOUIS V. (2002). *Conception et mise en oeuvre de modèles formels de calcul de plans d'action complexes par un agent rationnel dialoguant*. PhD thesis, Université de Caen/Basse-Normandie, France.

MCCARTHY J. & HAYES P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, volume 4, p. 463–502. Edinburgh University Press.

MEYER G., LOUIS V. & SANSONNET J.-P. (2005). Un opérateur de préférence adapté aux agents dialoguant. In Y. LESPÉRANCE & A. HERZIG, Eds., *MFI'05*, Caen, France : CEPADUES EDITION. A paraître.

RAO A. S. & GEORGEFF M. P. (1991). Modeling rational agents within a BDI-architecture. In *KR'91*, p. 473–484, San Mateo, CA, USA.

REITER R. (1980). A logic for default reasoning. *Artificial Intelligence*, **13**, 81–132.

ROSSI F., VENABLE K. B. & WALSH T. (2004). mCP nets : representing and reasoning with preferences of multiple agents. In *AAAI'04*, San Jose, CA, USA.

SADEK D. (1991). *Attitudes Mentales et Interaction Rationnelle : vers une théorie formelle de la Communication*. PhD thesis, Université de Rennes I, France.

SADEK D. (1992). A study in the logic of intention. In *KR'92*, p. 462–473. San Mateo, California.

VON WRIGHT G. (1972). The logic of preference reconsidered. *Theory and Decision*, **3**, 140–169.

Etude expérimentale de la coopération au sein d'un jeu à n joueurs

Thomas LALLART

Centre de Recherche en Informatique de Lens - CNRS
Université d'Artois 62300 LENS
lallart@cril.univ-artois.fr
http://www.cril.univ-artois.fr/~lallart/

Résumé :

Il existe encore peu d'études expérimentales de la coopération au sein des jeux à n joueurs. Pourtant, l'intérêt de ce type de problèmes dans le cadre des systèmes multi-agents, de la théorie des jeux et des sciences économiques est important.

Dans cet article, nous nous intéressons à l'étude des comportements des joueurs dans les jeux à n joueurs à somme nulle. Nous définissons un jeu très simple dans ce cadre et nous avons pour but d'identifier les bonnes stratégies, déterminer si des comportements coopératifs (coalitions) peuvent y être observés et si cette coopération est profitable aux joueurs. Nous souhaitons également étudier la dynamique de ces éventuelles coalitions. Cet article décrit quelques expérimentations effectuées sur ce jeu et dresse un bilan des premiers résultats obtenus.

Mots-clés : jeux à n joueurs, coopération, simulations, théorie des jeux computationnelle.

1 Introduction

En intelligence artificielle, les problèmes de coopération dans les systèmes multi-agents représentent un domaine de recherche important. Nous avons conçu un système d'étude de la coopération basé sur un jeu à n joueurs.

Dans le cadre des jeux, Axelrod (1984) a montré que des études expérimentales basées sur des simulations informatiques produisaient des résultats intéressants. Pour plus d'informations sur la théorie des jeux, le lecteur peut se référer à Von Neumann & Morgenstern (1944) et Luce & Raiffa (1947).

Arthur (1994) a montré via des simulations que des comportements simplement inductifs (basés sur l'utilisation des coups du passé uniquement) arrivent à faire émerger un comportement rationnel dans un jeu de minorité. Palmer *et al.* (1994) ont montré qu'à partir de comportements individuels simples, il est possible de faire *émerger* des comportements de groupe *rationnels* (au sens de la théorie économique) ; de nombreux modèles de marchés financiers basés sur des agents ont par exemple été développés et testés.

Nous nous proposons d'étudier la coopération et la formation de coalitions au sein d'un jeu. Nous avons nommé ce jeu *SADE* (*Simple Attack and Defense Environment*). Le jeu proposé est intéressant car les règles mettent en avant une possibilité de coopération entre joueurs bien qu'ils soient tous adversaires. Pour atteindre notre objectif, nous devons tout d'abord comparer plusieurs stratégies entre elles et découvrir quelles sont les meilleures. La théorie des jeux ne nous permet pas toujours d'obtenir des résultats concrets, ce qui est le cas pour le jeu proposé. Elle nous indique qu'il existe une solution (au moins un équilibre de Nash) mais elle ne nous dit pas comment la trouver. En particulier, les notions de *jeu sous forme coalitionnelle* (Weber (1994)) et de *structures de coalitions* (Greenberg (1994)) ne nous permettent aucunement de conclure sur ce type de jeu. De plus, il existe peu d'études sur les jeux à n joueurs sur lesquelles s'appuyer. Enfin, les recherches arborescentes classiques sont inefficaces car le facteur de branchement de ces algorithmes sur ce jeu est gigantesque.

Nous devons donc procéder à des expérimentations sur plusieurs stratégies. Nous les testons dans un environnement virtuel pour découvrir quels sont les comportements qui semblent être les plus efficaces. Nous cherchons également à déterminer si un comportement coopératif émerge et s'il est profitable aux joueurs. Les stratégies ont été testées sur des confrontations de deux puis de trois populations de stratégies. A l'issue des expérimentations, nous évaluons les nombres de victoires et de défaites de chaque stratégie. L'ensemble des résultats ainsi que le simulateur sont disponibles à l'URL `http://www.cril.univ-artois.fr/~lallart/sade/`.

Nous exposons dans cet article les règles du jeu et ses caractéristiques dans la section 2. Ensuite, nous expliquons comment sont réalisés les tests dans la section 3. Nous détaillons les stratégies qui ont été utilisées pour ces tests, les choix que nous avons faits et les restrictions que nous nous sommes imposées. Nous décrivons les tests qui ont été effectués puis nous présentons les résultats obtenus et les interprétations que nous avons pu en faire. Enfin, nous concluons par un bilan des résultats dans la section 4, puis nous exposons les travaux liés à l'étude réalisée dans la section 5 et les perspectives de développement dans la section 6.

2 Présentation du jeu

Chaque joueur possède un même nombre de ressources initiales. Le but du jeu est de récupérer toutes les ressources des autres joueurs. Pour augmenter ses ressources, le joueur doit réussir une attaque contre un autre joueur. Pour cela, il a l'opportunité de demander à autant de joueurs qu'il le souhaite de soutenir son attaque. Les joueurs ont la possibilité d'accepter ou de refuser de soutenir les attaques qu'on leur propose. Un coup se décompose donc en trois phases :

1. chaque joueur choisit quel joueur il attaque (exactement un joueur),
2. il choisit ensuite une liste de joueurs à qui il demande de soutenir son attaque,
3. ces joueurs acceptent ou refusent de soutenir cette attaque.

L'attaque est réussie si la somme des ressources du joueur attaquant et des joueurs qui ont accepté de soutenir l'attaque est strictement supérieure aux ressources du joueur attaqué. Si l'attaque réussit le joueur attaquant gagne un point et le joueur attaqué en perd

un sinon aucune ressource n'est perdue ou gagnée. Si un joueur n'a plus de ressource, il est éliminé. On suppose dans cet article que les joueurs refusent de soutenir des attaques contre eux-mêmes. Les coups sont joués de manière simultanée ; tous les joueurs décident et annoncent en même temps les joueurs qu'ils attaquent et ceux à qui ils demandent du soutien ainsi que les demandes de support qu'ils acceptent ou refusent. Il n'y a donc pas de phase de négociation pendant laquelle les joueurs peuvent discuter des possibilités qui s'offrent à eux et négocier des alliances. La partie s'arrête s'il ne reste plus qu'un seul joueur en jeu ou si les trois derniers tours du jeu sont identiques (c'est-à-dire les ressources des joueurs restent inchangées et tous les joueurs prennent les mêmes décisions).

On remarque que le jeu est à somme nulle : tout ce qui est gagné par un joueur est perdu par les autres. De plus, comme les coups sont joués de manière simultanée, le jeu est à information imparfaite. On remarque aussi que soutenir l'attaque d'un autre joueur ne bénéficie pas directement à celui qui donne son soutien (cela ne lui apporte pas de ressource). Cela sert seulement à montrer son soutien à un autre joueur (et, par exemple, d'en attendre la même chose en retour lors d'un prochain tour). C'est donc grâce à ces soutiens que l'on peut s'attendre à des phénomènes de coalitions.

Comme au premier tour tous les joueurs ont le même nombre de points, pour qu'une attaque réussisse il faut qu'au moins un joueur accepte de la soutenir. On peut aussi noter qu'une partie avec deux joueurs qui ont le même nombre de points ne se termine jamais (il est impossible de réussir une attaque).

Exemple 1
Soient 3 joueurs A, B et C à un instant quelconque de la partie. A possède 20 points, B 7 points et C 30 points. On remarque que C peut attaquer n'importe quel joueur avec succès. En effet, il a strictement plus de ressources que les autres, donc son attaque réussira. A peut attaquer B de la même façon ; son attaque réussira nécessairement. On remarque également que C est "intouchable". En effet, même si A et B s'allient contre C (c'est-à-dire A et B attaquent C, se demandent du soutien mutuellement et acceptent) ils ne cumuleront que 27 points de ressources ce qui n'est pas suffisant pour que l'attaque soit réussie. Donc C ne peut que gagner la partie. En effet, il est impossible pour lui de perdre des ressources et, de plus, à chaque tour il gagnera un point. En généralisant, on peut dire que si un joueur arrive dans une configuration de jeu dans laquelle une attaque de tous les autres joueurs (au moins 2) contre lui ne peut réussir, alors ce joueur a gagné la partie. S'il ne reste plus que 2 joueurs, alors c'est celui qui a strictement plus de ressources qui gagne.

Pour simplifier l'écriture et la compréhension, on dit qu'un joueur *attaque directement* un autre lorsqu'il le choisit comme joueur à attaquer. Un joueur *attaque indirectement* un autre lorsqu'il accepte de soutenir l'attaque d'un autre joueur contre celui-ci.

3 Expérimentations et résultats

3.1 Expérimentations réalisées

Notre but est d'avoir des premiers résultats avec des stratégies simples. Nous avons donc naturellement commencé par des tests simples. Nous nous sommes donc placés dans un espace de stratégie restreint (cf section 3.2) et les avons mises directement en

opposition sous la forme de duels. L'extension naturelle de cette expérimentation est de mettre en opposition trois stratégies différentes .

3.2 Stratégies utilisées

La stratégie du joueur définit la décision qu'il doit prendre à chaque étape du jeu. Elle peut prendre en compte l'état des joueurs (leurs ressources), les évènements passés, le temps écoulé depuis le début de la partie, etc. La stratégie d'un joueur peut se décomposer en 3 sous-stratégies, correspondant aux trois types de décisions qu'il doit prendre : la *sous-stratégie d'attaque*, la *sous-stratégie de demande de soutien* et la *sous-stratégie d'acceptation de soutien*. La première décide du joueur à attaquer, la seconde des joueurs à qui demander du soutien et la troisième des décisions de soutien à accepter ou à refuser.

Il existe principalement deux types de comportements simples pour les stratégies : prendre leurs décisions en fonction des ressources des joueurs ou en fonction des évènements passés. En ce qui concerne les évènements passés, on peut par exemple considérer :

- Les joueurs que j'ai déjà attaqué directement.
- Les joueurs qui m'ont déjà attaqués directement.

Et pour les ressources des joueurs, on peut par exemple considérer :

- Le joueur le plus fort.
- Le joueur le plus faible.
- Les joueurs plus forts que moi.
- Les joueurs moins forts que moi.

Considérons également les comportements simples qui ne prennent aucune information en compte pour décider : demander du soutien à personne / tout le monde et accepter toujours / jamais. En définissant des sous-stratégies à partir de ces comportements, on obtient 6 sous-stratégies d'attaque, 5 sous-stratégies de demande de soutien et 7 sous-stratégies d'acceptation de soutien. En construisant les stratégies issues de toutes les permutations possibles de ces sous-stratégies, on obtient 210 stratégies distinctes ; ce sont celles-ci qui sont utilisées lors des tests. Nous voulons tout d'abord effectuer des expérimentations sur des stratégies avec un comportement simple avant d'avancer vers des comportements plus complexes. En effet, les résultats obtenus par des expérimentations simples vont nous permettre de choisir plus judicieusement les stratégies et tests plus complexes que nous prévoyons de réaliser.

3.2.1 Sous-stratégies d'attaque

AttackAttackedPlayers (AtAP) : J'attaque les joueurs que j'ai déjà attaqués directement.
AttackStrongest (AtS) : J'attaque le joueur qui a le plus de ressources.
AttackStrongerThanMe (AtSTM) : J'attaque les joueurs qui ont plus de ressources que moi.
AttackWeakest (AtW) : J'attaque le joueur qui a le moins de ressources.
AttackWeakerThanMe (AtWTM) : J'attaque les joueurs qui ont moins de ressources que moi.
SpitefulAttack (AtSp) : J'attaque les joueurs qui m'ont déjà attaqués directement.

3.2.2 Sous-stratégies de demande de soutien

AskingEveryone (AsE) : Je demande du soutien à tous les joueurs.
AskingNobody (AsN) : Je ne demande du soutien à personne.
AskingPlayersDontAttackMe (AsPDAM) : Je demande du soutien aux joueurs qui ne m'ont pas attaqué directement.
AskingStrongerThanMe (AsSTM) : Je demande du soutien aux joueurs qui ont plus de ressources que moi.
AskingWeakerThanMe (AsWTM) : Je demande du soutien aux joueurs qui ont moins de ressources que moi.

3.2.3 Sous-stratégies d'acceptation de soutien

AcceptAskingPlayerStrongerThanMe (AcAsPSTM) : J'accepte de soutenir l'attaque si le joueur attaquant a plus de ressources que moi, je refuse sinon.
AcceptAskingPlayerWeakerThanMe (AcAsPWTM) : J'accepte de soutenir l'attaque si le joueur attaquant a moins de ressources que moi, je refuse sinon.
AcceptAttackedPlayerStrongerThanMe (AcAtPSTM) : J'accepte de soutenir l'attaque si le joueur attaqué a plus de ressources que moi, je refuse sinon.
AcceptAttackedPlayerWeakerThanMe (AcAtPWTM) : J'accepte de soutenir l'attaque si le joueur attaqué a moins de ressources que moi, je refuse sinon.
AcceptPlayersDontAttackMe (AcPDAM) : J'accepte de soutenir l'attaque si le joueur attaquant ne m'a pas attaqué directement, je refuse sinon.
AlwaysAcceptToSupport (AcA) : J'accepte toujours de soutenir l'attaque.
NeverAcceptToSupport (AcN) : Je n'accepte jamais de soutenir des attaques.

Si la sous-stratégie d'attaque ne permet pas de décider qui attaquer (par exemple, parce qu'elle désigne plusieurs joueurs) alors le joueur attaqué est choisi de façon pseudo-aléatoire. Le générateur pseudo-aléatoire n'est employé qu'à ce stade pour discriminer des joueurs ex-aequo. Pour simplifier l'écriture, on notera par X-Y-Z la stratégie composée de la sous-stratégie d'attaque X, de la sous-stratégie de demande de soutien Y et de la sous-stratégie d'acceptation de soutien Y. Notons par *-Y-Z l'ensemble des stratégies qui se composent d'une sous-stratégie d'attaque quelconque parmi celles que nous avons définies, de la sous-stratégie de demande de soutien Y et de la sous-stratégie d'acceptation de soutien Z.

3.3 Confrontation de deux stratégies (duels)

On met en confrontation une population composée de 10 stratégies A et de 10 stratégies B. Les joueurs commencent avec 20 points de ressources. Nous avons nommé ces tests *duels à 20 joueurs*. La confrontation des 210 stratégies dans des duels représente donc 21945 parties (chaque stratégie joue contre toutes les autres, nous avons donc $209 + 208 + \ldots + 2 + 1 = (210 * 209)/2 = 21945$ duels). Nous avons exécuté 4 fois ce test à partir de 4 graines d'initialisation des nombres pseudo-aléatoires différentes ce qui donne un total de 87780 parties. Ce test a été réitéré avec une population de 5 stratégies A et de 5 stratégies B. 87780 parties ont donc été jouées également. Nous avons nommé ces tests *duels à 10 joueurs*.

Dans beaucoup de cas, la partie ne se termine pas. Il peut y avoir plusieurs raisons à ce phénomène (aucune attaque ne réussit, toutes les attaques réussissent,etc.). C'est

pourquoi nous introduisons un nombre de tours maximal exécuté. On arrête la partie quand on atteint cette limite et elle est alors considérée comme nulle.

Les résultats sont présentés sous forme de tableaux. La colonne *Nb vict* indique le nombre de victoires de la stratégie. La colonne *Nb def* donne le nombre de défaites de la stratégie. La colonne *Nb nuls* donne le nombre de parties nulles de la stratégie. Pour les confrontations à trois joueurs, la colonne *Nb deux* indique le nombre de fois où la stratégie finit en seconde position. Enfin, la colonne *% vict* indique le taux de victoire de la stratégie par rapport au nombre de parties qu'elle a joué. Clairement, ce nombre X pour une stratégie A signifie : si un joueur joue avec la stratégie A alors il a $X\%$ de chances de gagner (dans l'environnement défini par le test).

Définir ce qu'est une "bonne" stratégie n'est pas chose aisée. Est-ce celles qui gagnent beaucoup ou celles qui perdent peu ? Nous regardons essentiellement le nombre de victoires ce qui ne constitue qu'une façon d'agréger les 3 nombres *Nb vict*, *Nb nuls* et *Nb def*. C'est pourquoi nous analysons aussi les défaites et les matchs nuls des stratégies.

3.3.1 Duels à 20 joueurs

Rang	Stratégie	Nb vict	Nb nuls	Nb def	%vict
1	AtWTM-AsE -AcA	618	14	204	73.92
	AtWTM-AsPDAM-AcPDAM	618	14	204	73.92
	AtWTM-AsPDAM-AcA	618	14	204	73.92
	AtWTM-AsE -AcPDAM	618	14	204	73.92
5	AtAP -AsE -AcA	566	198	72	67.70
	AtAP -AsE -AcPDAM	566	198	72	67.70
	AtAP -AsPDAM-AcPDAM	566	198	72	67.70
	AtAP -AsPDAM-AcA	566	198	72	67.70
	AtSp -AsE -AcA	566	198	72	67.70
	AtSp -AsPDAM-AcPDAM	566	198	72	67.70
	AtSp -AsPDAM-AcA	566	198	72	67.70
	AtSp -AsE -AcPDAM	566	198	72	67.70
13	AtW -AsPDAM-AcA	482	9	345	57.66
	AtW -AsE -AcA	482	9	345	57.66
	AtW -AsE -AcPDAM	482	9	345	57.66
	AtW -AsPDAM-AcPDAM	482	9	345	57.66
17	AtSTM -AsE -AcA	464	362	10	55.50
	AtSTM -AsPDAM-AcA	464	362	10	55.50
	AtSTM -AsE -AcPDAM	464	362	10	55.50
	AtSTM -AsPDAM-AcPDAM	464	362	10	55.50
21	AtS -AsE -AcPDAM	246	425	165	29.43
	AtS -AsE -AcA	246	425	165	29.43
	AtS -AsPDAM-AcPDAM	246	425	165	29.43
	AtS -AsPDAM-AcA	246	425	165	29.43

TAB. 1 – Les 24 premières stratégies des duels à 20 joueurs

Les 24 premières stratégies sont présentées dans le tableau 1 où l'on remarque immédiatement que seules les sous-stratégies de demande de soutien qui demandent du support à tout le monde (AsE) et à ceux qui ne l'ont pas attaqué (AsPDAM) et les sous-stratégies d'acceptation de soutien qui acceptent toujours (AcA) et qui acceptent si le joueur qui demande du support ne l'a pas attaqué (AcPDAM) sont représentées. A l'inverse, toutes les sous-stratégies d'attaque sont présentes et sont formées avec les 4 combinaisons possibles des sous-stratégies précédemment citées sous forme de "paquets". Les stratégies qui gagnent le plus souvent sont composées avec la sous-stratégie d'attaque AtWTM (attaquer les moins fort qu'elle) mais totalisent 204 défaites. On note

que les stratégies au rang 17 qui, bien qu'elles gagnent moins souvent (464 fois contre 618 pour les premières), ne perdent que 10 fois. Ces dernières ont le même comportement que celles qui gagnent le plus souvent si ce n'est qu'elles attaquent les joueurs les plus forts (AtSTM).

Les 19 stratégies qui ne gagnent jamais sont : AtS-AsN-*, AtSTM-AsN-*, AtAP-AsSTM-AcAsPSTM, AtSp-AsSTM-AcAsPSTM, AtSTM-AsSTM-AcAtPWTM, AtSp-AsSTM-AcN, AtAP-AsSTM-AcN. Les stratégies de la forme AtS-AsN-* et AtSTM-AsN-* ne gagnent jamais une partie. En effet, elles implémentent la négation des conditions de succès. Leur comportement est d'attaquer un joueur au moins aussi fort qu'elles sans demander du soutien : aucune attaque ne peut réussir.

Nous avons également identifié un groupe de 4 stratégies particulièrement robustes présentées dans le tableau 2. En effet, elles ne perdent jamais : elles gagnent ou font match nul. Ces stratégies ne gagnent que 138 fois mais ne sont jamais battues. Elles

Stratégie	Nb vict	Nb nuls	Nb def	%vict
AtSTM-AsE -AcAsPSTM	138	698	0	16.51
AtSTM-AsPDAM-AcAtWTM	138	698	0	16.51
AtSTM-AsE -AcAtWTM	138	698	0	16.51
AtSTM-AsPDAM-AcAsPSTM	138	698	0	16.51

TAB. 2 – Les stratégies qui perdent le moins dans les duels à 20 joueurs

attaquent les plus forts (AtSTM), demandent du soutien à tout le monde (AsE) ou à ceux qui ne les ont pas attaquées (AsPDAM) et acceptent si le joueur attaquant est plus fort qu'elles (AcAsSTM) ou si le joueur attaqué est moins fort qu'elles (AcAtWTM). Dès qu'un adversaire devient trop fort, ces stratégies vont l'attaquer souvent, l'empêchant ainsi de prendre trop d'avance. C'est pourquoi elles conduisent souvent à des matchs nuls.

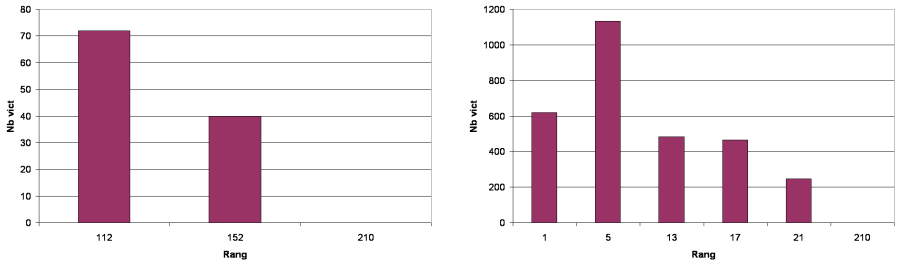
Nous avons calculé les taux de réussite des couples de sous-stratégies pour chaque combinaison possible. Les meilleurs groupes de stratégies sont présentés tableau 3.

Stratégie	Nb vict	% vict
* -AsE -AcPDAM	2942	58.65
-AsE -AcA	2942	58.65
-AsPDAM-AcPDAM	2942	58.65
-AsPDAM-AcA	2942	58.65
AtWTM-* -AcPDAM	1512	36.17
AtWTM-* -AcA	1512	36.17
AtWTM-AcPDAM-*	2014	34.42
AtWTM-AcE -*	2014	34.42

TAB. 3 – Les groupes de stratégies qui gagnent le plus dans les duels à 20 joueurs

Dans ce tableau, on observe que quelle que soit la façon dont une stratégie attaque, si elle demande du soutien avec AsE ou AsPDAM et qu'elle accepte de soutenir avec AcA ou AcPDAM, elle gagne en moyenne dans 58.65% des cas (c'est le meilleur pourcentage). Dans les duels, parmi les sous-stratégies proposées, les sous-stratégies de demande de soutien et d'acceptation de soutien semblent avoir plus d'importance que la sous-stratégie d'attaque pour gagner.

Dans la figure 1, nous avons représenté les rangs où se situent les victoires des stratégies dans le classement (en abscisse) et le nombre de victoires (en ordonnée) des groupes de stratégies AtAP-AsN-* et *-AsE-AcA. Le premier montre un groupe de stratégies que l'on peut qualifier de "mauvais". En effet, ces stratégies ne comptent que



(a) AtAp-AsN-* (b) *-AsE-AcA

FIG. 1 – Graphiques de groupes de stratégies.

peu de victoires ce qui les placent loin dans le classement. Les stratégies de ce groupe n'apparaissent d'ailleurs que 2 fois : au rang 112 avec 72 victoires et au rang 152 avec 40 victoires. A l'inverse, le second montre un groupe de stratégies qui gagnent souvent. C'est pourquoi les instances du groupe *-AsE-AcPDAM (c'est à dire quand on affecte une sous-stratégie d'attaque) se trouvent toutes au début du classement avec un grand nombre de victoires.

3.3.2 Duels à 10 joueurs

Les 24 premières stratégies sont représentées dans le tableau 4.

Rang	Stratégie	Nb vict	Nb nuls	Nb def	%vict
1	AtAP -AsE -AcA	584	168	84	69.86
	AtAP -AsE -AcPDAM	584	168	84	69.86
	AtAP -AsPDAM-AcPDAM	584	168	84	69.86
	AtAP -AsPDAM-AcA	584	168	84	69.86
	AtSp -AsE -AcA	584	168	84	69.86
	AtSp -AsPDAM-AcPDAM	584	168	84	69.86
	AtSp -AsPDAM-AcA	584	168	84	69.86
	AtSp -AsE -AcPDAM	584	168	84	69.86
9	AtWTM-AsE -AcA	567	0	269	67.82
	AtWTM-AsPDAM-AcPDAM	567	0	269	67.82
	AtWTM-AsPDAM-AcA	567	0	269	67.82
	AtWTM-AsE -AcPDAM	567	0	269	67.82
13	AtW -AsPDAM-AcA	534	0	302	63.86
	AtW -AsE -AcA	534	0	302	63.86
	AtW -AsE -AcPDAM	534	0	302	63.86
	AtW -AsPDAM-AcPDAM	534	0	302	63.86
17	AtSTM -AsE -AcA	476	332	28	56.94
	AtSTM -AsPDAM-AcA	476	332	28	56.94
	AtSTM -AsE -AcPDAM	476	332	28	56.94
	AtSTM -AsPDAM-AcPDAM	476	332	28	56.94
21	AtS -AsE -AcPDAM	243	428	165	29.07
	AtS -AsE -AcA	243	428	165	29.07
	AtS -AsPDAM-AcPDAM	243	428	165	29.07
	AtS -AsPDAM-AcA	243	428	165	29.07

TAB. 4 – Les 24 premières stratégies des duels à 10 joueurs

On constate que les 24 premières stratégies sont les mêmes que lors des duels à 20 joueurs à l'ordre près. En particulier, lors de ce test, les stratégies qui attaquent les plus faibles (AtWTM) finissent légèrement derrière celles qui attaquent les joueurs déjà

attaqués par la stratégie (AtAP) et ceux qui l'ont déjà attaquée (AtSp), alors qu'elles étaient en tête lors des duels à 20 joueurs. Le groupe des 4 stratégies au rang 17 (qui sont les mêmes que lors des duels à 20 joueurs) ne totalisent que 28 défaites par stratégie.

Les 26 stratégies qui ne gagnent jamais sont : AtS-AsN-*, AtSTM-AsN-*, AtAP-AsSTM-AcAtPWTM, AtAP-AsSTM-AcAsPSTM, AtAP-AsSTM-AcN, AtS-AsWTM-AcAsPWTM, AtS-AsWTM-AcAtPSTM, AtS-AsWTM-AcN, AtSTM-AsWTM-AcN, AtSTM-AsWTM-AcAtPSTM, AtSp-AsSTM-AcN, AtSp-AsSTM-AcAtPWTM, AtSp-AsSTM-AcAsPSTM, AtSTM-AsWTM-AcAsPWTM. On retrouve logiquement les stratégies de la forme AtS-AsN-* et AtSTM-AsN-*. On remarque aussi un grand nombre de stratégies qui attaquent les plus forts et demandent du soutien aux plus faibles. On peut donc supposer que, dans les duels, tenter de liguer les faibles contre les forts ne soit pas particulièrement efficace.

Lors des duels à 10 joueurs, nous avons pu également identifier un groupe de stratégies robustes qui perdent très peu présentées dans le tableau 5. Ces deux stratégies ne

Stratégie	Nb vict	Nb nuls	Nb def	%vict
AtSTM-AsPDAM-AcAtPWTM	134	692	10	16.03
AtSTM-AsE -AcAtPWTM	134	692	10	16.03

TAB. 5 – Les stratégies qui perdent le moins lors des duels à 10 joueurs

perdent que 10 fois. Elles font également partie des stratégies qui ne perdent jamais lors des duels à 20 joueurs.

Stratégie	Nb vict	% vict
* -AsE -AcPDAM	2988	59.57
-AsE -AcA	2988	59.57
-AsPDAM-AcPDAM	2988	59.57
-AsPDAM-AcA	2988	59.57
AsE -* -AcPDAM	1384	33.11
AsE -* -AcA	1384	33.11
AtSp-* -AcPDAM	1384	33.11
AtSp-* -AcA	1384	33.11

TAB. 6 – Les groupes de stratégies qui gagnent le plus lors des duels à 10 joueurs

Les meilleurs groupes de stratégies sont présentés dans le tableau 6. Comme lors des duels à 20 joueurs, le groupe de tête se compose de AsE ou AsPDAM comme sous-stratégie de demande de soutien et de AcA ou AcPDAM comme sous-stratégie d'acceptation de soutien. Ce résultat renforce l'hypothèse que, pour ces duels, la sous-stratégie d'attaque importe moins que les deux autres sous-stratégies.

3.4 Confrontation de trois stratégies (truels)

Dans ce test, on met en confrontation 3 joueurs avec 3 stratégies différentes. Ce test représente 1565410 parties et a été exécuté pour 4 graines d'initialisation des nombres pseudo-aléatoires différentes. Au total 6261640 parties ont donc été jouées. Nous avons nommé ces tests *les truels*.

Les 16 meilleures stratégies sont présentées dans le tableau 7.

Comme pour les duels, les meilleures stratégies demandent du soutien à tout le monde (AsE) ou aux joueurs qui ne les ont pas attaqués (AsPDAM) et acceptent toujours de soutenir les attaques (AcA) ou acceptent si le joueur qui demande le soutien ne les a pas attaqués (AcPDAM). On retrouve toutes les sous-stratégies d'attaque sauf AtW et

Rang	Stratégie	Nb vict	Nb Deux	Nb def	Nb nuls	%vict
1	AtSTM-AsE -AcA	33445	7897	16028	32082	37.39
2	AtSTM-AsE -AcPDAM	33207	8284	15884	32077	37.12
3	AtSTM-AsPDAM-AcA	33151	8520	15843	31938	37.06
4	AtSTM-AsPDAM-AcPDAM	33032	8775	15704	31941	36.92
5	AtAP -AsE -AcA	31866	22162	8866	26558	35.62
6	AtAP -AsPDAM-AcA	31821	22376	8821	26434	35.57
7	AtAP -AsE -AcPDAM	31733	22528	8729	26462	35.50
8	AtAP -AsPDAM-AcPDAM	31568	22800	8747	26337	35.29
9	AtSp -AsE -AcA	31336	22602	9031	26483	35.03
10	AtSp -AsE -AcPDAM	30893	23144	9045	26370	34.54
11	AtSp -AsPDAM-AcA	30843	22903	9252	26454	34.48
12	AtSp -AsPDAM-AcPDAM	30280	23503	9329	26340	33.85
13	AtS -AsE -AcA	28717	5498	13022	42215	32.10
14	AtS -AsE -AcPDAM	28607	5498	13186	42161	31.98
15	AtS -AsPDAM-AcA	28497	5222	13449	42284	31.86
16	AtS -AsPDAM-AcPDAM	28396	5219	13607	42230	31.74

TAB. 7 – Les 16 premières stratégies des confrontations de 3 stratégies

AtWTM qui sont un peu plus loin dans le classement. On peut noter que les 4 meilleures stratégies attaquent les joueurs les plus forts (AtSTM).

Les stratégies qui perdent le moins sont présentées dans le tableau 8.

Rang	Stratégie	Nb vict	Nb Deux	Nb def	Nb nuls	%vict
1	AtW-AsE -AcA	23734	39619	1806	24293	26.53
2	AtW-AsE -AcPDAM	24159	39197	1811	24285	27.01
3	AtW-AsPDAM-AcA	23982	39160	2025	24285	26.81
4	AtW-AsPDAM-AcPDAM	24294	38774	2107	24277	27.16

TAB. 8 – Les 5 stratégies qui perdent le moins dans les confrontations à 3 joueurs

Ce tableau montre que les stratégies qui perdent le moins attaquent le plus faible (AtW), demandent du soutien à tous les joueurs (AsE) ou à ceux qui ne les ont pas attaquées (AsPDAM) et acceptent toujours de soutenir les attaques (AcA) ou acceptent si le joueur qui demande le soutien ne les a pas attaquées (AcPDAM). Ces résultats semblent montrer que, lors des truels, il vaut mieux attaquer les plus forts si on veut gagner mais qu'il vaut mieux attaquer le plus faible si on ne veut pas perdre.

Enfin, nous avons également effectué des statistiques sur les couples de sous-stratégies les plus efficaces lors de ces truels ; les meilleures sont présentées dans le tableau 9. Comme lors des duels, le meilleur groupe de stratégies ne tient pas compte de la sous-

Rang	Stratégie	Nb vict	Nb Deux	Nb def	Nb nuls	%vict
1	* -AsE -AcA	176216	131957	52834	175705	32.83
2	* -AsE -AcPDAM	176087	132788	52463	175374	32.81
3	* -AsPDAM-AcA	175847	132110	53289	175466	32.76
4	* -AsPDAM-AcPDAM	175449	132932	53189	175142	32.69
5	AtSTM-AsPDAM-*	194841	27535	59468	344320	31.12
6	AtSTM-AsE -*	193314	25989	61978	344883	30.87
7	AtAP -AsE -*	190254	86882	42992	306036	30.38
8	AtAP -AsPDAM-*	189462	88462	42176	306064	30.26

TAB. 9 – Les groupes de stratégies qui gagnent le plus dans les truels

stratégie d'attaque. Cependant, les résultats sont nettement plus modérés. En effet, ce groupe gagne dans près de 60% des cas dans les duels et seulement 32.83% lors des truels. Pour ce type d'affrontement, nous ne pouvons donc pas supposer que la sous-stratégie d'attaque importe moins que les deux autres sous-stratégies.

Il y a 65 stratégies qui ne gagnent jamais une partie lors de ces truels. En particulier, on trouve AtS-AsN-*, AtS-AsSTM-*, AtSTM-AsN-* et AtSTM-AsSTM-*. Ce résultat était prévisible : AtS-AsN-* et AtSTM-AsN-* ne peuvent pas gagner de part la construction de la stratégie et AtS-AsSTM-* et AtSTM-AsSTM-* ont peu de chance de réussir des attaques. En effet, ces stratégies attaquent les plus forts et leur demandent du soutien (qui sera refusé car elles ne supportent pas les attaques contre elles-mêmes).

4 Bilan

Les résultats les plus marquants sont la présence de seulement 2 sous-stratégies de demande de soutien (AsE et AsPDAM) et 2 sous-stratégies d'acceptation de soutien (AcA et AcPDAM) parmi la composition des meilleures stratégies des confrontations à 2 et à 3 joueurs. Les stratégies de la forme *-AsPDAM-AcPDAM ont un comportement que l'on peut qualifier de rancunier. En effet, dès que ces stratégies ont été attaquées par un joueur, elles ne lui demanderont plus de soutien et n'accepteront plus de le soutenir. De plus, AtSp-AsPDAM-AcPDAM, qui est la stratégie la plus rancunière que l'on puisse construire, est dans le groupe de tête des 3 tests. Ces stratégies coopèrent tant qu'on ne les a pas attaquées. Ce type de comportement semble efficace dans nos tests.

En ce qui concerne les sous-stratégies d'attaque pour les duels, elles se trouvent toutes à un niveau équivalent et ce sont les deux autres sous-stratégies qui font la différence. On peut donc fortement supposer que, dans l'environnement des tests effectués, les sous-stratégies de demande de soutien et d'acceptation de soutien jouent un rôle plus important que la sous-stratégie d'attaque. Ce résultat n'est plus vrai pour les truels.

Nous avons pu identifier également des stratégies robustes qui sont difficiles à battre. Dans les duels, elles sont toutes composées de la sous-stratégie d'attaque AtSTM, de la sous-stratégie de demande de soutien AsE ou AsPDAM et de la sous-stratégie d'acceptation de soutien AcAsPSTM ou AcAtPWTM. Le trait commun est donc qu'elles attaquent les joueurs les plus forts de la partie. Ce type de comportement à tendance à équilibrer les forces. En effet, quand un joueur devient trop fort, ces stratégies vont l'attaquer jusqu'à ce qu'il s'affaiblisse à leurs niveaux. On peut supposer que, lors de cette phase, des coalitions peuvent avoir été formées. A l'inverse, lors des truels, nous avons vu que les stratégies qui perdent le moins attaquent les plus faibles et que celles qui gagnent le plus souvent attaquent les plus forts. En effet, statistiquement, les stratégies de la forme AtSTM-*-* gagnent dans 13.33% des cas et perdent dans 17.25% des cas alors que celles de la forme AtWTM-*-* gagnent dans 12.25% des cas mais perdent seulement dans 8.84% des cas. Nous pouvons donc dire que, dans les confrontations à 3 joueurs réalisées, un bon moyen de gagner est d'attaquer les plus forts et que pour ne pas perdre il vaut mieux attaquer les plus faibles. Il semble que ce soit l'inverse dans les duels. Dans les deux cas, la sous-stratégie de demande de soutien doit être AsE ou AsPDAM.

Nous avons également répertorié les stratégies qui ne gagnent jamais. On retrouve toujours AtS-AsN-* et AtSTM-AsN-*. Ce résultat est logique puisque ces deux stratégies ne réussiront jamais une attaque (elles implémentent la négation des conditions de succès d'une attaque). De plus, attaquer les joueurs les plus forts peut être dangereux car on risque de les fâcher.

D'après les premiers résultats, il semble que nous devons retenir AsE et AsPDAM comme "bonnes sous-stratégies" de demande de soutien et AcA et AcPDAM comme "bonnes sous-stratégies" d'acceptation de soutien. D'après la définition du jeu, il semblait assez évident que les joueurs devaient demander du soutien aux autres pour gagner. Cette intuition s'est confirmée avec les résultats obtenus. En effet, les deux sous-stratégies de demande de soutien que nous avons retenues sont AsE et AsPDAM. La première demande du soutien à tous les joueurs et on peut supposer que la seconde demande du soutien à suffisamment de joueurs pour que les attaques soient régulièrement réussies. De même, pour les acceptations de soutien, les sous-stratégies retenues sont AcA et AcPDAM. La première accepte toujours de soutenir les attaques et la seconde accepte si le joueur qui lui demande ne l'a pas attaqué. Ces comportements sont une première forme de coopération : les stratégies demandent du soutien et acceptent d'aider les autres régulièrement. Nous avons donc montré que, dans les tests réalisés, les meilleures stratégies n'agissent pas seules et indépendamment des autres mais demandent de l'aide et acceptent d'aider les autres régulièrement. Il semble donc que la coopération soit bénéfique aux joueurs (au moins dans l'environnement des tests). Cette idée est confortée par le fait que les stratégies résolument non coopératives (qui ne demandent jamais de soutien ou refusent toujours d'aider) sont en bas du classement. Nous ne savons cependant pas encore si cette coopération est durable. Lorsque des joueurs s'allient, nous n'avons pas d'éléments qui nous permettent de prouver qu'ils continuent à coopérer aux tours suivants.

Ces résultats montrent que notre jeu est cohérent, qu'il semble constituer une bonne base de travail pour notre étude sur la formation de coalitions et que nous pouvons donc continuer nos expérimentations avec des stratégies et des tests plus complexes.

5 Travaux liés

Des recherches sur les jeux à partir d'expérimentations ont été réalisés sur le dilemme itéré du prisonnier par Beaufils (2000) et Delahaye & Mathieu (1996). Les techniques d'évaluations de stratégies utilisées sont inspirées de ces travaux. Cependant, les études sur le dilemme itéré du prisonnier ne permettent pas d'étudier la coopération en coalition ; il n'y a que deux joueurs.

Yao & Darwen (1994) décrivent des expérimentations sur le dilemme itéré du prisonnier à n joueurs. Les tests sont réalisés par une évolution écologique basée sur un algorithme génétique. Ce type de test permet d'avoir des résultats plus robustes car l'environnement dans lequel les stratégies s'affrontent évolue à chaque génération. Il existe des points communs entre le dilemme itéré du prisonnier à n joueurs et *SADE*. En effet, il n'existe pas de phase de négociation dans les deux jeux. Les joueurs ne peuvent donc pas communiquer explicitement mais ils peuvent le faire de manière implicite par les coups joués. On peut imaginer par exemple une stratégie qui, lorsqu'elle accepte de supporter une attaque d'un joueur A contre elle-même, envoie en fait un message pour annoncer à d'autres qu'elles doivent attaquer A ensemble au tour suivant.

Le jeu *Diplomacy* peut également être comparé à *SADE*. Il s'agit d'un jeu de stratégie où l'on déplace des armées sur une carte d'Europe. Il existe quelques travaux sur la programmation de strategies pour *Diplomacy* (Hall & Loeb (1992) et Kraus & Lehmann

(1995)), avec des résultats mitigés. *SADE* peut être considéré comme une abstraction de ce jeu, ou l'on supprime les aspects spatiaux pour se concentrer sur la confrontation de joueurs possédant des ressources. Trouver de bonnes stratégies pour *SADE* pourrait donc être un premier pas vers la conception de bonnes stratégies pour *Diplomacy*.

6 Perspectives

Après avoir obtenu des résultats sur des stratégies simples, nous pouvons entamer des tests plus complexes. En particulier, nous prévoyons de réaliser des duels en faisant varier la proportion des stratégies mises en jeu. Plutôt que d'opposer x stratégies A contre x stratégies B, nous opposons x stratégies A contre $(nbJ - x)$ stratégies B où $x \in 1, .., nbJ - 1$ et $nbJ > 2$ et nous pouvons faire varier x.

Les sous-stratégies dont les décisions prennent en compte des évènements passés possède un paramètre appelé *mémoire*. Il représente le nombre de tours pendant lequel la sous-stratégie mémorise les évènements survenus à un certain tour de la partie. Lorsqu'une stratégie mémorise les évènements depuis le début du jeu, elle a une *mémoire infinie*, ce qui est le cas des stratégies utilisées lors de nos expérimentations. Cependant, il peut être préjudiciable pour une stratégie de mémoriser trop longtemps les informations. Lors des prochains tests, nous souhaitons également évaluer l'importance du paramètre mémoire des sous-stratégies et son influence sur les résultats des parties.

Pour que les stratégies soient les plus souples possibles, nous avons introduit un mécanisme de filtrage. Chaque sous-stratégie est en réalité composée d'une liste ordonnée de sous-stratégies (que l'on nomme filtres). Par exemple, soient 3 filtres d'attaque A, B et C placés dans cet ordre dans la liste. Cette liste prend en entrée l'ensemble des joueurs encore en jeu. Si le filtre A retourne strictement plus d'un joueur alors cette sous-liste est envoyée au filtre B. Le filtre B effectue la même opération. S'il y a toujours plus d'un joueur alors la sous-liste est envoyée au filtre C. Cette opération de filtrage est effectuée jusqu'à ce qu'il n'y ait plus qu'un seul joueur dans cette liste : c'est celui qui sera attaqué. Pour la sous-stratégie de demande de soutien, c'est l'intersection des filtres qui la composent qui est retournée. Enfin pour la sous-stratégie d'acceptation de soutien, il faut que chaque filtre qui la compose accepte de soutenir l'attaque pour qu'elle soit supportée. De cette façon, nous pouvons construire des stratégies de manière beaucoup plus fine et leur donner un comportement plus complexe à partir de filtres simples.

Maintenant que des résultats ont été obtenus à partir de stratégies simples, nous pouvons comparer les stratégies que nous avons qualifiées de "bonnes" avec des stratégies plus complexes. En particulier, nous souhaitons mettre en jeu des stratégies qui prennent en compte les acceptations et les refus de soutien des autres joueurs. En effet, les demandes de soutien sont un moteur de coopération.

Afin d'analyser les coalitions formées par les joueurs, nous devons analyser en détails le déroulement des parties. Définir quelles conditions nous permettent de dire que deux joueurs se trouvent dans une même coalition n'est pas trivial. Nous pouvons imaginer plusieurs définitions. Par exemple, deux joueurs sont dans une même coalition si :

1. Ils ne se sont pas attaqués directement.
2. Ils ne se sont pas attaqués ni directement ni indirectement.

3. Ils ne se sont pas attaqués ni directement ni indirectement, se sont mutuellement demandés du soutien pour leurs attaques et ont accepté de s'aider.
4. Ils ne se sont pas attaqués ni directement ni indirectement, se sont mutuellement demandés du soutien pour leurs attaques, ont accepté de s'aider et ont attaqué le même joueur.

Ces 4 définitions possibles constituent une graduation de la notion de coalition.

Enfin, nous cherchons à construire des parties se comportant un peu à la manière des oscillateurs des automates cellulaires ; des parties où des tours identiques apparaissent successivement tous les n tours, donc infinies. Nous avons déjà des parties infinies où il ne se passe rien (aucune attaque ne réussit jamais), mais nous aimerions trouver des exemples où les attaques réussissent et donc où les joueurs gagnent et perdent des points successivement.

Nous maintenons un site web qui présente l'ensemble des résultats obtenus à l'adresse `http://www.cril.univ-artois.fr/~lallart/sade/`.

Remerciements

L'auteur remercie l'Université d'Artois, la Région Nord/Pas-de-Calais et le FEDER pour leur support. L'auteur tient également à remercier les relecteurs pour leurs remarques sur la version préliminaire de cet article.

Références

ARTHUR B. (1994). Inductive reasoning and bounded rationality : the el-farol problem. *American Economic Review*, **84**, 406–417.

AXELROD R. (1984). *The Evolution of Cooperation*. Basic Books.

BEAUFILS B. (2000). *Modèles et simulations informatiques des problèmes de coopérations entre agents*. PhD thesis, Université des Sciences et Technologies de Lille.

DELAHAYE J.-P. & MATHIEU P. (1996). *Experiences sur le Dilemme Itéré des Prisonniers*. Rapport interne 233, Université des Sciences et Technologies de Lille.

GREENBERG J. (1994). *Handbook of Game Theory*, chapter 37 - Coalitional structures, p. 1306–1333. Elsevier Science.

HALL R. & LOEB D. (1992). Thoughts on programming a diplomat. In *Heuristic programming in Artificial Intelligence 3 : the third computer olympiad*, p. 123–145.

KRAUS S. & LEHMANN D. (1995). Designing and building a negociating automated agent. In *Computational Intelligence 11*, p. 132–171.

LUCE D. & RAIFFA H. (1947). *Games and Decisions*. Dover.

PALMER R., ARTHUR W., HOLLAND J., LEBARON B. & TAYLER P. (1994). Artificial economic life : A simple model of a stockmarket. *Physica D*, **75**, 264–274.

VON NEUMANN J. & MORGENSTERN O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.

WEBER R. J. (1994). *Handbook of game theory*, chapter 36 - Games in coalitional form, p. 1286–1302. Elsevier Sciences.

YAO X. & DARWEN P. J. (1994). An experimental study of n-person iterated prisoner's dilemma games. In *Evo Workshops*, p. 90–108.

Articles courts

Session Poster

Dynamiques de marchés et comportements d'agents

Julien DERVEEUW

Laboratoire d'Informatique Fondamentale de Lille - UMR USTL/CNRS 8022
Equipe Systèmes Multi Agents et Comportements (SMAC)
Groupe Complexité, Interaction Stratégique et Coopération (CISCO)
Université des Sciences et Technologies de Lille
59655 Villeneuve d'Ascq Cédex
julien.derveeuw@lifl.fr http://www.lifl.fr/~derveeuw

Récemment, la simulation de marchés financiers à l'aide d'agents hétérogènes est devenue un domaine actif, sous l'impulsion, notamment, du *Santa Fe Institute*. A la suite des travaux de Arthur (1994) et LeBaron (1995), qui montrent qu'avec des comportements individuels simples, il est possible d'obtenir des comportements globaux rationnels, de nombreux modèles de marchés financiers ont été développés. Ces modèles ont pour but de reproduire et comprendre des phénomènes économiques (bulles et krachs par exemple). Le *Santa Fe Artificial Stock Market* (SF-ASM) est une référence majeure dans ce domaine. Il a été montré avec ce modèle qu'il était possible d'obtenir un comportement global rationnel au sens économique avec des agents qui apprennent uniquement de leurs expériences passées.

Le principe du SF-ASM est simple : des agents possèdent des actions du même type. L'action a un *prix* courant et rapporte une somme d'argent appellée *dividende*. Les agents doivent prendre une décision : investir leur argent dans de nouvelles actions ou vendre celles qu'ils possèdent pour récupérer leur mise. Comme la dynamique des prix est soumise à l'influence de la loi de l'offre et de la demande, les décisions prises par les agents ont un impact direct sur le prix de l'action. Les agents peuvent également investir leur argent dans un capital sans risque qui rapporte un interêt fixe.

Le SF-ASM a été modifié de nombreuses fois pour corriger quelques aspects techniques mineurs du modèle original. Bien que ce modèle soit imparfait, nous essayons de le corriger pour obtenir un modèle adapté à l'étude de l'influence des comportements individuels des agents sur la dynamique globale du prix.

Nos travaux sont directement basés sur les articles de Palmer *et al.* (1994) et de LeBaron (2002). Dans ces travaux, les décisions prises par les agents (acheter ou vendre) sont aléatoires : même si cette prise de décision est motivée par un processus d'apprentissage, rien ne garantit que la décision finale aura une quelconque logique économique. Nous corrigeons dans un premier temps ce processus de prise de décision pour donner plus de sens aux choix faits par les agents.

Pour prendre leurs décisions, les agents observent des indicateurs de marché. Ceux-ci, dans le modèle original, donnent des informations sur les tendances (à la hausse ou à

la baisse) et sur le prix que l'action devrait avoir dans un système économique rationnel. Comme l'un de nos buts est de comprendre les moteurs de la dynamique de marché, nous créons deux types d'agents qui utilisent des classes d'indicateurs différentes. Les agents du groupe n'utilisant que des indicateurs provenant de la théorie économique classique seront appellés *agents fondamentalistes* et ceux du groupe n'utilisant que des indicateurs techniques des *agents spéculateurs*. Cette séparation nous permet d'étudier la dynamique de marché en fonction de la proportion des agents fondamentalistes et spéculateurs.

1 Processus de raisonnement des agents

Dans le SF-ASM, les agents ne connaissent pas le processus de génération du dividende et du prix. Ils doivent élaborer des stratégies en utilisant uniquement leur expérience (les valeurs passées du prix et du dividende). Leur objectif est de prendre des décisions qui maximisent leur satisfaction (leur richesse). Ici, ce comportement est vu comme *essayer de reconnaître un état particulier du marché* pour ensuite *essayer de prendre la meilleure décision possible en fonction de cet état*. Plus précisement, chaque agent a un stock de m règles qui décrivent des états du marché et qui lui disent quelle décision prendre. Une règle est composée de trois sous-parties : des *conditions*, une *action* et une *force*. Par exemple, la composition de la k-ième règle d'un agent a_i est la suivante :

1. les *conditions* peuvent être vues comme une chaîne (*chromosome*) composée de symbôles $\{\texttt{0},\texttt{1},\#\}$. Chaque caractère (*gène*) contribue à la description d'une condition sur le marché. Ces indicateurs peuvent être `faux` (`0`), `vrai` (`1`) ou `non significatif` (#). Par exemple, un exemple de condition peut être "*Le prix de l'action est au dessus de 200€*".
2. l'*action* $a_{i,k}$ qui représente la décision que l'agent doit prendre si la k-ième règle est sélectionnée. Cette action est indépendante de la condition.
3. une *force* $s_{i,k}$ qui décrit l'efficacité de la règle à avoir fait gagner de l'argent dans le passé.

Pour maximiser leurs gains, les agents essayent de prendre la meilleure décision pour l'itération courante. Ils commencent par sélectionner dans leur stock de règles celles dont tous les indicateurs sont cohérents avec l'état courant du marché. Parmi ces règles sont choisies celles avec une force $s_{i,k} > 0$. Une des règles est ensuite gardée grâce à un processus de sélection aléatoire proportionelle à leur force. La partie *action* de cette règle donne la décision de l'agent pour le jour courant. Si aucune règle n'a été sélectionnée par ce processus, on considère que l'agent ne dispose pas de connaissances suffisantes pour bien appréhender le jour courant de marché et il prend donc la décision de ne rien faire.

Périodiquement, un algorithme génétique est lancé pour améliorer le stock de règles de chaque agent. Les pires règles (celles avec la force la plus petite) sont d'abord supprimées. Elles sont remplacées par de nouvelles règles à l'aide d'un processus génétique classique : les meilleures règles (avec la plus grande force) sont sélectionnées pour

être les parents de nouvelles règles en utilisant un processus de sélection proportionnel à leur force. Les nouvelles règles sont ensuite générées soit par mutation soit par crossing-over.

2 Nos deux types d'agents

Dans le SF-ASM, rien ne garantit que les décisions des agents aient une quelconque logique économique. Nous essayons donc d'améliorer le modèle des agents en introduisant une telle logique dans leurs règles. Nous définissons pour cela le concept de *rationnalisation* d'une règle qui, en fonction de ses *conditions*, calcule une action économiquement rationnelle. Le modèle d'agent original ne permettant pas d'étudier l'influence des comportements individuels des agents sur la dynamique globale du prix, nous séparons les agents en deux populations distinctes. Les agents *fondamentalistes* étudient le rapport entre la valeur fondamentale et le prix de l'action alors que les agents *spéculateurs* étudient les tendances courantes du marché (prix à la hausse, à la baisse).

Les agents fondamentalistes sont l'incarnation de la théorie économique classique : si une action est sur-côtée (son prix est supérieure à sa valeur fondamentale), l'agent essaye de la vendre et, symétriquement, essaye d'acheter une action sous-côtée. Leur procédure de rationnalisation est le reflet de cette logique : les indicateurs permettent de déterminer un encadrement du rapport entre le prix et la valeur fondamentale qui sert à déterminer s'il faut que l'agent achète, vende ou ne fasse rien.

Les agents spéculateurs essayent de réaliser un profit maximum en suivant la tendance courante du marché grâce à des indicateurs spécifiques : moyennes mobiles, comparaison entre le prix courant et extrema de prix locaux, etc. Ainsi, si la tendance générale du marché est à la hausse, un comportement rationnel pour les agents spéculateurs est d'acheter de nouvelles actions et inversement. La procédure de rationnalisation détermine l'action de la règle en observant la tendance donnée par la majorité des indicateurs. Comme les agents spéculateurs ne possèdent pas d'indicateurs observant la valeur fondamentale de l'action, ils peuvent spéculer de manière infinie et faire (dé)croître le prix sans limite. Pour remédier à ce problème, nous ajoutons au modèle d'agent spéculateur une notion de *peur* (ou *risque*). En effet le but pour un spéculateur n'est pas de détenir des actions mais d'en acheter le plus possible tant que le prix monte puis de s'en débarasser avant que les prix diminuent. Si les échanges deviennent trop faibles sur le marché, le spéculateur risque de ne plus pouvoir se séparer du stock d'actions qu'il a achetées durant une période à la hausse du marché. Le degré de liquidité du marché (nombre d'actions échangées à chaque itération) détermine donc la limite de spéculation : tant que le marché est liquide (et donc qu'il est possible de se séparer de revendre des actions), le spéculateur spécule, mais dès qu'une limite basse est atteinte, il prend peur et essaye de revendre les actions en sa possession avant que cela ne devienne impossible. Dans notre modèle, chaque agent dispose de sa propre appréciation de ce seuil en dessous duquel il risque de ne plus pouvoir se séparer des actions qu'il possède.

3 Premiers résultats

Nous étudions l'influence des agents spéculateurs et des agents fondamentalistes sur la dynamique globale du prix de l'action. Quand la population est uniquement composée de fondamentalistes, le prix de l'action reste près de la valeur fondamentale. On retrouve ici les mêmes résultats que dans Palmer *et al.* (1994) : des agents en rationnalité limitée sans aucune connaissance du processus de génération de la valeur fondamentale arrivent à s'y adapter. Cependant, nos agents conservent un comportement économique rationnel d'après la théorie économique classique ce qui n'était pas le cas dans Palmer *et al.* (1994). De plus, la dynamique de prix engendrée par les agents est une marche aléatoire et est valide dans un contexte économique : ces prix pourraient être ceux d'un marché réel (à efficience faible).

Symétriquement, nous testons les effets d'une population composée uniquement par des agents spéculateurs. Cela amène à une dynamique de prix avec des variations de grande amplitude : les agents spéculateurs spéculent jusqu'à ce qu'ils aient atteint leur limite de risque, et inversent ensuite leur comportement. Les prix dans ce cas sont facilement prédictibles et sont quasiment cycliques. Le marché n'est plus efficient dans ce cas caricatural.

Nous avons également étudié l'influence des populations mixtes composées à la fois de spéculateurs et de fondamentalistes sur la dynamique des prix. On remarque que les agents fondamentalistes ont un effet régulateur sur les autres agents. Alors que les spéculateurs représentent la majeure partie de la population, la dynamique de prix reste une marche aléatoire et donc une dynamique de prix valide pour la théorie économique. De plus, nous pouvons remarquer un nouveau phénomène qui n'était pas présent dans Arthur et al. : de grandes variations entre le prix et la valeur fondamentale apparaissent assimilables à des *bulles* et des *krachs*. Nous réussissons à faire émerger sans biais une dynamique de groupe proche de phénomènes observables sur des marchés financiers réels.

Références

ARTHUR B. (1994). Inductive reasoning and bounded rationality : the el-farol problem. *American Economic Review*, **84**, 406–417.

LEBARON B. (1995). Experiments in evolutionnary finance. *Working Paper, University of Wisconsin - Madison*.

LEBARON B. (2002). Building the santa fe artificial stock market. *Working Paper, Brandeis University*.

PALMER R., ARTHUR W., HOLLAND J., LEBARON B. & TAYLER P. (1994). Artificial economic life : A simple model of a stockmarket. *Physica D*, **75**, 264–274.

LoTREC:
Un Démonstrateur Générique par Tableaux

Mohamad Sahade

Institut de Recherche en Informatique de Toulouse,
118, route de Narbonne
31062 Toulouse Cedex 4
sahade@irit.fr

Résumé : Dans cet article, nous présentons un démonstrateur automatique générique pour les logiques modales et les logiques de description, nommé LoTREC, qui repose sur la méthode des tableaux. Il s'adresse aux chercheurs en logique utilisateurs de logiques modales, et aux étudiants concernés par l'apprentissage de ces logiques. LoTREC génère des modèles et des contre-modèles pour des formules modales, ces modèles sont présentés à travers une interface graphique. En premier lieu, nous donnons une introduction aux systèmes de tableaux, ensuite nous présentons le langage simple mais expressif servant à définir des logiques et des stratégies de recherche, sur lequel LoTREC repose. Enfin, nous terminons par la conclusion. **Mots-clés** : Démonstration automatique, Logique modale, Méthodes de tableaux.

1 Introduction

Les logiques modales sont très populaires et apparaissent sous divers aspects dans beaucoup de domaines de l'informatique, y compris la représentation des connaissances, le domaine des logiques des programmes et l'informatique linguistique. Les logiques modales interviennent aussi dans la formalisation de l'interaction entre agents rationnels, plus spécifiquement dans la formalisation de plusieurs aspects relatifs à la sécurité des applications Internet (authentification de signature, autorisation). Le problème de base qui doit être résolu dans la plupart des applications des logiques modales est celui de la satisfiabilité des formules. Il y a des différentes approches à ce problème, incluant les méthodes de traduction, et les systèmes de preuve basés sur la méthode des tableaux. De plus, ces approches nécessitent la mise au point de procédures automatiques, ainsi que la possibilité de tests et d'expérimentations.

Les dernières années ont vu de nombreux systèmes efficaces basés sur la méthode des tableaux (Horrocks & Patel-Schneider, 1999; Horrocks *et al.*, 2000). Cependant, les comparaisons qui se sont tenues aux ateliers de logiques et aux conférences de TABLEAUX ont montré un problème important : l'importance accordée à la rapidité est si forte que la plupart des implémentations sont limitées à des logiques fixes. Cependant,

il y a une infinité de logiques modales et le choix d'une logique ou d'une autre se fait en accord avec les besoins et les contraintes informatiques de ses applications. Même avec la même logique, différentes stratégies de recherche peuvent être nécessaires pour différentes applications. Si un utilisateur veut utiliser des logiques ou même rechercher des stratégies légèrement différentes de ceux des systèmes courants, il doit coder son propre démonstrateur.

Pour répondre aux besoins des utilisateurs souhaitant expérimenter et travailler avec différentes logiques ou stratégies, on a besoin d'un démonstrateur générique de théorème jouant le même rôle qu'Isabelle (Paulson, 1994) ou PVS (Owre *et al.*, 1992) pour les logiques d'ordre supérieur, tout en étant moins complexe. Si l'utilisateur n'est pas la même personne que le programmeur du démonstrateur, on a besoin de :1. flexibilité et portabilité de l'exécution, 2. langages de haut niveau pour la définition des règles de tableau et stratégie, 3. interfaces faciles à utiliser.
LoTREC répond à ces exigences et il vise à couvrir toutes les logiques ayant la sémantique des mondes possibles, en particulier les logiques modales et les logiques de description. LoTREC est non seulement un démonstrateur pour résoudre le problème de satisfiabilité des formules modales, mais il génère également des modèles et contre modèles pour des formules logiques. Une première version de LoTREC a été implémenté par D. Fauthoux (Farinas del Cerro *et al.*, 2001) qui ne permettait pas d'implémenter des propriétés sémantiques comme la linéarité, la confluence, ni de tester par exemple si une formule a été réalisée dans un noeud. Récemment, il a été modifié par M. Sahade et une nouvelle version est désormais exécutable via la Toile(Sahade *et al.*, 2005).

2 LoTREC

2.1 Principes de Base

Les tableaux sont représentés en général sous forme d'arbres, mais avec LoTREC ils sont généralisés en graphes pour permettre l'implémentations de logiques plus complexes comme par exemple les logiques de la confluence, ou les logiques d'interaction entre connaissances et actions. Les graphes permettent aussi de construire des modèles de Kripke c.à.d des modèles basés sur la sémantique des mondes possibles. Dans LoTREC les noeuds sont étiquetés par des formules, et les arcs par des termes qui permettent de représenter des relations d'accessibilités composées. LoTREC présente les tableaux qu'il génère par des graphes qu'on peut restructurer et modifier pour obtenir un meilleur affichage. Il permet aussi selon l'avis de l'utilisateur d'obtenir un encodage des tableaux dans des fichiers, pour une éventuelle interaction avec d'autres démonstrateurs. LoTREC a une interface graphique pour la définition de connecteurs, de règles, de stratégies, et des formules à tester.

Le but de flexibilité, portabilité, et d'interfaces nous a motivé à choisir JAVA comme langage d'implémentation. Avec un tel langage orienté objet, LoTREC utilise une architecture basée sur les événements. Pour l'utiliser il faut définir la logique en définissant les connecteurs, les règles de tableaux et la stratégie de recherche.

2.2 Définir sa Logique

Dans LoTREC, la description d'une logique est constituée de deux parties : des connecteurs et des règles de tableau. Ces deux derniers sont définis dans le langage spécifique de LoTREC.

2.2.1 Connecteurs

Pour déterminer le langage d'une logique, il faut définir ses connecteurs. Pour définir un connecteur dans LoTREC il faut utiliser la syntaxe suivante :
`connector Nom_Int Nb_Arg Asso "Ext_Pres" P`
Où `Nom_Int` est le nom interne du connecteur, `Nb_Arg` est le nombre d'arguments, `Asso` est une variable booléen qui prend la valeur true si le connecteur est associatif et la valeur false si non. `Ext_Pres` est la manière d'afficher le connecteur et ses arguments. `P` est la priorité (0,1,2,...) du connecteur par rapport aux autres. Plus le nombre est grand, plus la priorité est importante.

2.2.2 Règles

Comme dans toute méthode des tableaux sémantiques, les règles de tableau dans LoTREC décrivent l'evolution du tableau. Une règle de tableau est constituée de deux parties : " conditions" et " actions ". La partie " conditions " contient les conditions nécessaires pour que la règle soit applicable et la partie " actions " décrit les opérations qui devront s'effectuer si la règle est applicable. Le syntaxe des règles est :

`rule` *rule_name*
`if` *condition1*
⋮
`if` *conditionN*
`do` *action1*
⋮
`do` *actionM*
`end`

Pour une liste des descripteurs et des actions voir (Sahade *et al.*, 2005).

2.3 Définir sa Stratégie de Recherche

Après avoir défini les règles de tableau de la logique, la question qui se pose est comment les combiner et les appliquer. C'est la stratégie de recherche qui fait ce genre de travail. Elle met en correspondance des tableaux avec d'autres en appliquant répétitivement les règles de manière appropriée. Dans ce cas, appliquer une règle signifie "appliquer la règle simultanément sur tous les instances possibles dans le tableau".

La définition d'une stratégie se fait selon la grammaire BNF suivante :

GlobalStrategy : :=	`strategy`	NameOfStrategy	Block	`end.`
Block : :=	rule |			
	rule Block |			
	`repeat`	Block		`end` |
	`firstRule`	Block		`end` |
	`allRules`	Block		`end.`

Nous utilisons `firstRule rule1,...,ruleN end` pour appliquer la première règle applicable parmi `rule1,...,ruleN` ; nous utilisons `allRules rule1,...,ruleN end` pour appliquer toutes les règles applicables parmi `rule1,...,ruleN` dans l'ordre ; et `repeat rule1, ...,ruleN end` pour répéter l'application de toutes les règles applicables parmi `rule1, ..., ruleN`.

3 Conclusion

LoTREC est en cours d'amélioration, parmi les buts envisagés il y a : (1) la possibilité de communiquer avec d'autres démonstrateurs comme KSAT (Davis & Putnam, 1960) , (2) la possibilité de l'utiliser d'une façon interactive semi-automatique où l'utilisateur peut à un moment donné intervenir pour choisir l'application de telle ou telle règle, rediriger la recherche, supprimer des branches qui ne sont pas intéressantes. LoTREC est exécutable via l'Internet à l'adresse suivante :
`www.irit.fr/ACTIVITES/LILaC/Lotrec/`. Il offre aussi un tutorial, une librairie des logiques et des stratégies qui commencent par les logiques simples comme CPL, K,... jusqu'à des logiques complexes comme KDB, S4, TLL, PDL, logiques de densité, de confluence.

Références

DAVIS M. & PUTNAM H. (1960). A computing procedure for quantification theory. In *Journal of the ACM 7*, p. 201–205.

FARINAS DEL CERRO L., FAUTHOUX D., GASQUET O., HERZIG A., LONGIN D. & MASSACCI F. (2001). Lotrec : the generic tableau prover for modal and description logics. In *International Joint Conference on Automated Reasoning*, LNCS, p.6̃ : Springer Verlag.

HORROCKS I. & PATEL-SCHNEIDER P. F. (1999). Optimizing description logic subsumption. In *J. Log. Comput.*, volume 9,3, p. 267–293.

HORROCKS I., SATTLER U. & TOBIES S. (2000). Practical reasoning for very expressive description logics. In *cs.LO/0005013*.

OWRE S., RUSHBY J. M. & SHANKAR N. (1992). PVS : A prototype verification system. In *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, p. 748–752, Saratoga, NY : Springer-Verlag.

PAULSON L. C. (1994). Isabelle - a generic theorem prover (with a contribution by t. nipkow). In *Lecture Notes in Computer Science*, volume 828 : Springer.

SAHADE M., GASQUET O. & HERZIG A. (2005). Lotrec : Logical tableaux research engineering compagnion (manuel d'utilisation).

Opérationalisation des ontologies OWL dans la famille SG

Frédéric Comte, Michel Leclère

LIRMM, Université de Montpellier II,
161, rue Ada, F-34392 Montpellier cedex - France
{frcomte,leclere}@lirmm.fr

1 Introduction

Le web sémantique (Berners-Lee *et al.*, 2001) propose une nouvelle architecture pour le web basée autour de trois concepts : ressources, métadonnées et ontologies. On y considère le web comme un espace dont chaque ressource est un point repéré par un identifiant unique (un URI). Des métadonnées sont associées à ces points et servent à caractériser le contenu de la ressource qu'ils identifient. La sémantique de ces métadonnées est spécifiée grâce à des ontologies de domaine. Pour représenter métadonnées et ontologies, le W3C recommande des langages d'échange basés sur une syntaxe XML : RDF (Ressource Description Framework) et OWL (Ontology Web Language). Bien que disposant de sémantiques formelles, les langages proposés ne possèdent pas de services inférentiels propres. Pour raisonner sur les métadonnées et ontologies qu'ils permettent de représenter, on transforme ces représentations dans des formalismes disposant de mécanismes opérationnels de raisonnement. Actuellement le formalisme le plus utilisé est celui des logiques de description.

Pour notre part, nous nous intéressons au formalisme des graphes conceptuels, un formalisme de représentation des connaissances fondé sur la théorie logique et la théorie des graphes. Ce formalisme nous semble intéressant dans le cadre du web sémantique car il dispose d'une syntaxe intuitive pour un utilisateur non familier avec les formules logiques et qu'il permet d'envisager des scénarios d'utilisation des représentations basés sur une interaction graphique entre l'utilisateur et le système : lors de la création de métadonnées en permettant la présentation graphique d'incohérences, ou lors du traitement des requêtes, en permettant un affichage des réponses sous la forme de graphes annotations dont les sommets permettent d'accéder aux ressources qu'ils représentent. RDF est d'ailleurs un langage de triplets présentables sous forme de graphes.

Suivant cet objectif, nous étudions des transformations possibles des directives OWL dans un système formel particulier de graphes conceptuels issu de la famille SG (Baget & Mugnier, 2002). Une première caractérisation d'un sous ensemble du langage OWL, nous permet d'établir une correspondance entre les sémantiques formelles des deux formalismes. Notre objectif à terme est de baliser un sous langage OWL (que nous pourrions appelé OWL SG) le plus large possible pour lequel un système formel à base

de graphes conceptuels permettrait de mener des raisonnements corrects.

2 La famille SG

Les graphes conceptuels simples (SG) (Chein & Mugnier, 1992), les constructions de base du formalisme, ont les caractéristiques suivantes que nous considérons comme essentielles : (1) les connaissances sont représentées sous la forme de graphes étiquetés, qui décrivent des entités et des relations entre elles, (2) les raisonnements y sont effectués à l'aide d'une opération de morphisme de graphes particulière appelée projection, (3) les SGs sont fondés logiquement (la projection est adéquate et complète par rapport à la déduction en logique du premier ordre sur les formules conjonctives existentielles).

Dans (Baget & Mugnier, 2002) une famille d'extension des SG (la famille SG) est proposée pour représenter des règles et des contraintes. La caractéristique commune aux extensions proposées est qu'elles sont définies à partir de graphes simples et que les mécanismes de raisonnement restent basés sur la projection. Dans ce cadre, les graphes simples sont utilisés pour représenter des faits, des requêtes, ou comme blocs pour des constructions plus complexes comme les règles ou les contraintes.

Un graphe simple (en fait un multigraphe biparti) est composé de deux types de nœuds : les *nœuds concepts* qui représentent les entités du domaine et les *nœuds relations* qui représentent les relations entre ces entités. Tous ces nœuds sont étiquetés. Les étiquettes sont choisies parmi un vocabulaire conceptuel, appelé support, composé d'un ensemble ordonné de types primitifs de concepts (ayant le type $\top$ comme supremum), de la définition d'ensembles de types de concepts incompatibles entre-eux, d'un ensemble ordonné de types de relation ayant une arité fixée et d'un ensemble de marqueurs individuels. Un nœud concept est étiqueté par un type conjonctif acceptable (un ensemble de types n'incluant pas un ensemble de types incompatibles) (Chein & Mugnier, 2004), et soit un marqueur individuel soit le marqueur générique (représentant un individu indéfini). Un nœud relation est étiqueté par un type de relation et ses arêtes incidentes sont ordonnées de 1 à l'arité du type de la relation. Des liens de coréférence (arêtes particulières entre nœuds concepts) peuvent être ajoutés pour indiquer l'égalité entre deux entités. Un exemple de graphe simple est donné dans la figure 1.

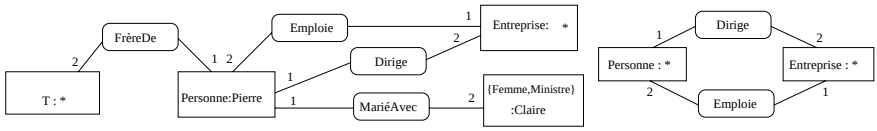


FIG. 1 – Exemple de graphe simple

Une λ-abstraction n-aire est obtenue à partir d'un graphe simple par l'identification de n sommets concepts génériques. Des couples de λ-abstractions de même arité permettent de représenter des règles. Une règle $R = (A, B)$ s'interpréte : "Si A est présent alors B peut être ajouté". La règle de la figure 2 indique que deux entités liées par la relation *MariéAvec* à une troisième sont identiques.

FIG. 2 – Règle spécifiant la caractéristique fonctionnelle de la relation *MariéeAvec*.

3 Un noyau pour OWL SG

Pour définir notre transformation, nous nous appuyons sur la syntaxe abstraite d'OWL sur laquelle est définie la sémantique principale du langage (Schneider *et al.*, 2004). L'objectif étant la réalisation d'inférences sur les représentations, nous nous focalisons sur les représentations dont la sémantique doit être prise en compte dans les raisonnements, délaissant les directives d'annotations, de gestion de version...

Le noyau d'OWL SG permet de représenter des faits (directive *Individual*) (sans utilisation de restrictions), de spécifier des hiérarchies de classes et de propriétés (axiomes *Class, SubClassof, DisjointClasses, ObjectProperty* et *SubPropertyOf*), et de spécifier des contraintes globales sur les propriétés (axiome restrictions *Domain* et *Range* et caractéristiques *Symmetric, Transitive, InverseOf, Functional, InverseFunctional*).

Nous ne prenons pas en compte la distinction entre les propriétés liant des classes (*ObjectProperty*) et les propriétes liant des classes à des types de données (*DatatypeProperty*) considérant que nous pouvons traiter les types de données comme des classes et les littéraux comme des individus.

Les faits

individual=Individual(*individualID***?** type(*classID*)***** value(**value**)*****)

value=*PropertyID individualID* | *PropertyID* **individual**

Chaque individu est transformé en un sommet concept étiqueté par le type conjonctif constitué des classID déclarés comme *type*, et soit par un marqueur individuel correspondant à l'individualID, soit par le marqueur générique. Chaque *value* associée est transformée en un sommet relation étiqueté par PropertyID dont les arêtes incidentes lient les individus concernés. Le fait *Individual(Pierre type(Personne) value(frèreDe individual()) value(mariéeAvec Individual(Claire type(Femme) type(Ministre))) value(dirige individual(type(Entreprise) value(emploie Pierre))))* donne le graphe gauche de la figure 1.

À la différence de OWL DL, on peut accepter dans OWL SG des cycles ne comportant que des sommets concepts génériques. Cela revient à autoriser l'utilisation de variables (attribut nodeID de RDF) pour désigner un individu. Ainsi, le fait *Individual(_ :x type(Personne) value(dirige Individual(type(Entreprise) value(emploie _ :x))))* donne le graphe droit de la figure 1.

Les hiérachies de classes

axiome=Class(*classID* partial *classID******) | SubClassOf(*classID classID*) | DisjointClasses(*classID classID classID******) | ObjectProperty(*PropertyID* super(*PropertyID*)*** contrainte***) | SubPropertyOf(PropertyID PropertyID)

La transformation associe à chaque classe (resp. chaque propriété) un type de concept (resp. un type de relation binaire). Les axiomes indiquent l'ordre des hiérarchies de types du support ou des types incompatibles. Le type prédéfini owl :Thing est transformé en $\top$, le type de concept supremum de la hiérarchie des types de concept.

Les contraintes globales des propriétés

contrainte=inverseOf(*PropertyID*)**?** | Symmetric **?** | **[**Functional | InverseFunctional | Functional InverseFunctional | Transitive **] ?** | domain(*classID*) | range(*classID*)

La transformation associe à ces contraintes des règles. La figure 2 présente la règle correspondant à la directive *Functional* associée à la propriété *MariéAvec*, la figure 3 présente les règles correspondant à *Transitive* associée à la propriété *FrèreDe* et *domain(Personne) range(Personne)* associés à la propriété *mariéAvec*.

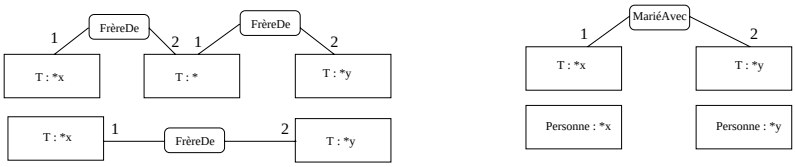


FIG. 3 – Les règles correspondant à *Transitive*, *domain* et *range*

4 Conclusion

OWL Lite et DL ont été choisis pour être opérationalisés à l'aide des logiques de description. De la même façon, nous définissons un sous langage OWL SG destiné à être opérationalisé avec la famille SG.

Nous avons proposé une transformation pour un noyau minimal de ce sous-langage qui préserve la sémantique formelle (Schneider *et al.*, 2004). Cette transformation permet de disposer d'un système formel de raisonnement graphique adéquat et complet permettant de traiter des requêtes existentielles conjonctives.

La prise en compte d'autres axiomes comme *EquivalentClasses*, *SameIndividual* et *EquivalentProperties* ne posent pas de problèmes majeurs mais nécessitent de modifier légèrement le formalisme de la famille SG (prise en compte de préordre dans le support, relâchement de l'hypothèse du nom unique (UNA) pour les marqueurs individuels...). Enfin, les restrictions telles que *someValuesFrom*, *allValuesFrom*, et les *cardinalités* devraient pouvoir en partie être représentées en utilisant le mécanisme des règles. C'est le travail que nous poursuivons actuellement.

Références

BAGET J.-F. & MUGNIER M.-L. (2002). The complexity of rules and constraints. *JAIR*, **16**, 425–465. http ://www.jair.org/abstracts/baget02a.html.

BERNERS-LEE T., HENDLER J. & LASSILE O. (2001). The semantic web. *Scientific American*.

CHEIN M. & MUGNIER M.-L. (1992). Conceptual Graphs : Fundamental Notions. *Revue d'Intelligence Artificielle*, **6**(4), 365–406. Available at http ://www.lirmm.fr/˜mugnier/.

CHEIN M. & MUGNIER M.-L. (2004). Types and coreference in simple conceptual graphs. In *ICCS'04*, volume 3127/2004 of *LNAI*, p. 303 – 318 : Springer.

SCHNEIDER P. P., HAYES P. & HORROCKS I. (2004). *OWL Web Ontology Language Semantics and Abstract Syntax*. Recommandation, W3C. http ://www.w3.org/TR/owl-semantics/.

Projection d'une ontologie dans un espace muni d'une mesure sémantique

Emmanuel Blanchard

Laboratoire d'informatique de Nantes Atlantique
École polytechnique de l'université de Nantes
rue Christian Pauc
BP 50609 - 44306 Nantes Cedex 3
emmanuel.blanchard@univ-nantes.fr

Résumé : Les ontologies sont au coeur du processus de gestion des connaissances. De nombreux champs d'application nécessitant la gestion de connaissances cherchent à exploiter la richesse des ontologies : le web sémantique, l'interopérabilité des systèmes, la recherche d'information, la gestion des compétences en entreprise, le e-learning, le traitement automatique du langage naturel, etc. Des travaux définissent des mesures sémantiques basées sur des ontologies lexicales, comme WordNet, pour identifier le sens d'un mot. Nous proposons de décontextualiser ces mesures dans le cadre d'une étude sur la projection d'une ontologie dans un espace muni d'une mesure sémantique.

Mots-clés : Ingénierie des connaissances, ontologies, mesures sémantiques.

1 Introduction

Un large consensus est maintenant établi concernant la définition et le rôle des ontologies en ingénierie des connaissances : « an ontology is a formal, explicit, specification of a shared conceptualization » (Gruber, 1993). Les ontologies constituent des référentiels de connaissances supportant l'échange d'informations entre systèmes informatiques. De nombreux champs d'application cherchent à exploiter leur richesse : le web sémantique, l'interopérabilité des systèmes, la recherche d'information, la gestion des compétences en entreprise, le e-learning, le traitement automatique du langage naturel, etc.

Des travaux proposent des mesures sémantiques basées sur des ontologies (souvent des ontologies lexicales comme WordNet). Nous proposons de décontextualiser ces mesures dans le cadre d'une étude sur la projection d'une ontologie dans un espace muni d'une mesure sémantique. Ce nouvel espace de connaissances nous semble être un support privilégié pour l'exploitation d'une ontologie. Dans la suite de cet article, nous analysons quelques mesures sémantiques dans l'optique d'une projection de l'ontologie restreinte aux liens de subsomption.

2 Les mesures sémantiques

Dans notre approche, nous considérons les primitives de base de toute ontologie que sont les concepts et les relations (Fürst, 2004). S'il existe de nombreuses relations possibles, certaines d'entre elles sont utilisées de façon plus ou moins récurrente. Par exemple, les relations taxonomiques (hyperonymie/hyponymie) qui correspondent au lien de subsomption (is-a) sont les relations de base commune à quasiment toutes les ontologies. C'est pourquoi nous avons fait le choix de concentrer nos réflexions sur ces relations avant de tenter de généraliser cette étude à d'autres types de relations.

La validation des mesures est évoquée selon trois angles (Budanitsky, 1999) : *l'analyse mathématique* : examination théorique des propriétés mathématiques d'une mesure ; *la comparaison avec le jugement humain* : analyse de la corrélation avec les évaluations subjectives de sujets humains ; *l'évaluation dans une application spécifique* : expérimentation dans un cadre applicatif donné. Lorsqu'il ne s'agit pas de valider une mesure, mais de comparer plusieurs mesures entre elles, l'examination théorique préalable nous semble indispensable. On trouve dans la littérature les termes de *distance sémantique* (semantic distance), *similarité sémantique* (semantic similarity) et *connectivité sémantique* (semantic relatedness) entre deux concepts d'une même ontologie. La *similarité sémantique* évalue *la ressemblance entre deux concepts*. La *connectivité sémantique* évalue *la proximité entre deux concepts*. La similarité sémantique est un cas particulier de connectivité sémantique. La *distance sémantique* évalue *l'éloignement entre deux concepts*. Elle est la notion inverse de celle de connectivité sémantique. De plus, la mesure inverse de la similarité est également une distance sémantique.

Dans la suite, nous présentons huit mesures de manière à faciliter leur comparaison. Les différentes primitives suivantes sont utilisées pour exprimer leur formule : $rt \rightsquigarrow$ racine de l'ontologie ; $cpts \rightsquigarrow$ ensemble des concepts de l'ontologie ; $pths(x, y) \rightsquigarrow$ ensemble des chemins entre les concepts x et y ; $len(x) \rightsquigarrow$ longueur en nombre d'arcs du chemin x ; $P(x) \rightsquigarrow$ probabilité d'apparition d'un concept x dans un corpus ; $mscs(x, y) \rightsquigarrow$ concept le plus spécifique subsumant x et y.

Mesure de Rada, Mili, Bicknell et Blettner. La distance sémantique proposée par Rada et ses collègues (Rada *et al.*, 1989) est basée sur la considération du plus court chemin entre deux concepts dans une ontologie.

$$\forall c_i, c_j \in cpts, \quad dist_{rmbb}(c_i, c_j) = \min_{p \in pths(c_i, c_j)} len(p) \tag{1}$$

Mesure de Resnik. La mesure de similarité de Resnik (Resnik, 1995) repose sur l'hypothèse que plus deux concepts partagent d'information en commun, plus ils sont similaires. Sur la base de la théorie de l'information, l'auteur propose de considérer le contenu informationnel des concepts : $CI(c) = -\log(P(c))$. $P(c)$ correspond à la probabilité d'occurrence, dans un corpus de texte conséquent, du concept c ou de l'un des concepts qu'il subsume directement ou indirectement. L'information partagée par deux concepts est alors égal au contenu informationnel de leur subsumant commun le plus spécifique.

$$\forall c_i, c_j \in cpts, \quad sim_r(c_i, c_j) = -\log P(mscs(c_i, c_j)) \tag{2}$$

Mesure de Wu et Palmer. Dans leur travaux, Wu et Palmer (Wu & Palmer, 1994) définissent une similarité entre concepts dans une ontologie restreinte aux liens taxonomiques. Les deux paramètres que sont la taille des chemins de c_i à $mscs(c_i, c_j)$ et de c_j à $mscs(c_i, c_j)$ de la formule de Wu et Palmer ont été remplacés par celui du plus court chemin entre c_i et c_j dans la formule présentée.

$$\forall c_i, c_j \in cpts,$$
$$sim_{wp}(c_i, c_j) = \frac{2 * \min\limits_{p \in pths(mscs(c_i,c_j),rt)} len(p)}{\min\limits_{p \in pths(c_i,c_j)} len(p) + 2 * \min\limits_{p \in pths(mscs(c_i,c_j),rt)} len(p)} \tag{3}$$

Mesure de Jiang et Conrath. Une autre distance sémantique a été proposée par Jiang et Conrath (Jiang & Conrath, 1997) qui ont utilisé comme Resnik, un corpus en plus de l'ontologie. Ils expriment la distance entre deux concepts par la différence entre le contenu informationnel des deux concepts et le contenu informationnel de leur subsumant commun le plus spécifique.

$$\forall c_i, c_j \in cpts,$$
$$dist_{jc}(c_i, c_j) = 2 * \log P\Big(mscs(c_i, c_j)\Big) - \Big(\log P(c_i) + \log P(c_j)\Big) \tag{4}$$

Mesure de Lin. La mesure la mieux analysée par son auteur est celle de Lin (Lin, 1998) qui propose une mesure de similarité sémantique basée sur une ontologie et un corpus. Cette similarité tient compte de l'information partagée par les deux concepts à la manière de P. Resnik, mais aussi de ce qui les distingue. On retrouve les deux composantes présentes dans la mesure de Jiang et Conrath mais au lieu de faire leur différence, Lin fait leur rapport.

$$\forall c_i, c_j \in cpts,$$
$$sim_l(c_i, c_j) = \frac{2 * \log P\Big(mscs(c_i, c_j)\Big)}{\Big(\log P(c_i) + \log P(c_j)\Big)} \tag{5}$$

Synthèse. La mesure de Leacock et Chodorow (Leacock & Chodorow, 1998), celle de Hirst et St Onge (Hirst & Onge, 1998) comme celle de Sussna (Sussna, 1993) entre dans la cadre de cette étude. Leur formule peut également être exprimée en partie avec certaines des primitives utilisées précédemment. Après une première analyse des mesures, nous avons dégagé leurs caractéristiques communes : *sources d'information :* toutes les mesures considérées se basent sur une ontologie (souvent WordNet). Certaines utilisent un corpus de textes en complément de l'ontologie. Ce sont les deux seules sources d'informations exploitées par les mesures étudiées ; *principes :* la plupart des mesures proposées reposent sur des principes clairement explicités par leurs auteurs comme le contenu informationnel ou le plus court chemin ; *classe sémantique :* chaque mesure a sa propre sémantique, mais on peut identifier sa classe sémantique (distance, similarité ou connectivité sémantique)

Chaque mesure prend en compte différents éléments de l'ontologie ou du corpus et les combine à sa manière selon certains principes. Au final, chaque mesure est influencée

par deux types de paramètres que sont la taille des plus courts chemins entre certains concepts et la densité des concepts sur ces chemins.

3 Conclusion

Dans un premier temps, nous avons proposé une étude décontextualisée de huit mesures dans un cadre général de projection d'une ontologie dans un espace muni d'une mesure sémantique. Une liste des caractéristiques essentielles des mesures sémantiques qui se dégagent de cette étude a été établie.

Concernant l'utilisation d'un corpus en complément d'une ontologie, il nous semble qu'elle apporte peu d'informations supplémentaires pour un coût algorithmique relativement conséquent. La visualisation d'ontologie sur la base de mesures sémantiques apparaît prometteuse pour l'aide au consensus lors du développement d'une ontologie ainsi que pour la maintenance des ontologies.

Références

BUDANITSKY A. (1999). *Lexical semantic relatedness and its application in natural language processing*. Rapport interne, Conputer Systems Research Group - University of Toronto.

FÜRST F. (2004). *Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationalisation*. PhD thesis, Ecole polytechnique de l'université de Nantes.

GRUBER T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, **5**(2), 199–220.

HIRST G. & ONGE D. S. (1998). Lexical chains as representation of context for the detection and correction of malaproprisms. In C. FELLBAUM, Ed., *WordNet : An electronic lexical database*, p. 305–332. MIT Press.

JIANG J. J. & CONRATH D. W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of International Conference on Research in Computational Linguistics*.

LEACOCK C. & CHODOROW M. (1998). Combining local context and wordnet similarity for word sense identification. In C. FELLBAUM, Ed., *WordNet : An electronic lexical database*, p. 265–283. MIT Press.

LIN D. (1998). An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, p. 296–304 : Morgan Kaufmann.

RADA R., MILI H., BICKNELL E. & BLETTNER M. (1989). Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, **19**(1), 17–30.

RESNIK P. (1995). Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 1, p. 448–453.

SUSSNA M. (1993). Word sense disambiguation for free-text indexing using a massive semantic network. In *Proceedings of the Second International Conference on Information and Knowledge Management*, p. 67–74.

WU Z. & PALMER M. (1994). Verb semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting of the Associations for Computational Linguistics*, p. 133–138.

Un modèle computationnel biomimétique de navigation pour le robot-rat Psikharpax

Alexandra d'Erfurth[1], Adrien Peyrache[1], Agnès Guillot[1] et Angelo Arleo[2]

[1] AnimatLab-LIP6, CNRS/Université Paris 6 ; [2] Neuroscience Group, SONY CSL, Paris
Correspondance à : alexandra.derfurth@lip6.fr

Résumé : L'objectif de ce travail est de proposer un modèle computationnel de navigation accordant à un robot-rat biomimétique la capacité d'apprendre en parallèle plusieurs stratégies de navigation lui permettant de rejoindre un but dans un environnement inconnu. Le modèle préliminaire déjà mis en œuvre connecte un modèle de l'hippocampe avec un modèle de colonnes corticales préfrontales attribuant à un robot simulé des stratégies locales et des stratégies de planification.

Mots-clés : modélisation computationnelle biomimétique, réseaux neuronaux, représentation de l'espace, navigation, planification, robotique autonome.

1 Introduction

Cet article présente un travail effectué dans le cadre du projet Psikharpax[1] visant à créer un robot-rat biomimétique, autonome et adaptatif (Meyer et al., 2005). Nous disposons actuellement d'une architecture de contrôle intégrant un modèle de sélection de l'action, dont le fonctionnement est basé sur des connaissances neurobiologiques précises, et un modèle de navigation (Girard et al., sous presse). L'objectif de ce travail est de remplacer ce dernier modèle par une architecture biomimétique constituée de modules différents représentant l'hippocampe et le cortex préfrontal médial. Ceux-ci fourniront respectivement au système de sélection de l'action des stratégies de navigation réactives locales –utilisant une carte basée sur des références externes à l'agent (Redish, 1999)– et des stratégies cognitives de planification vers un but –utilisant une carte basée sur les mouvements de l'agent (Ethier et al. 2001). Ces stratégies concurrentes pourront être sélectionnées spécifiquement en fonction de certains indices présents ou non dans l'environnement –ce qui a été démontré chez le rat (de Bruin et al., 2001). On suppose que cette capacité accordera une meilleure adaptabilité à un robot mobile qui n'aurait pas accès à tous les indices spatiaux pertinents pour rejoindre les buts nécessaires à la réalisation de sa tâche.

[1] Projet mené en collaboration avec le Laboratoire de Physiologie et de la Perception de l'Action du Collège de France, l'Adaptive Behaviour Research Group de l'Université de Sheffield et les compagnies BEV et Wany S.A..

2 Description du modèle et résultats

Le modèle de navigation - en cours de construction - connecte un modèle hippocampique (Arleo et al., 2004) à un modèle de colonnes corticales préfrontales inspiré de celui de Bieszczad & Pagurek (1997). Le premier élabore une carte constituée de « cellules de lieu » dont chaque activation est corrélée à une zone spatiale occupée par l'agent, sans représentation de transitions entre ces zones. Le second élabore une carte de ces transitions construite à partir des mouvements réalisés. Elle repose sur la création dynamique de colonnes corticales à partir de la carte hippocampique. Des transitions apprises entre ces colonnes construisent une carte topologique de l'environnement permettant la recherche et l'exécution de chemins conduisant aux buts mémorisés. Nous allons nous limiter ici à la description du fonctionnement de ce dernier modèle.

L'organisation en colonnes du cortex est une propriété très récemment reprise dans les modèles biomimétiques de planification (Koene & Hasselmo, sous presse). Chaque colonne est créée au cours de l'exploration et connectée –par apprentissage Hebbien– à un ensemble de cellules de lieux actives en même temps pour une position donnée de l'agent. Ainsi, le modèle effectue un *changement d'échelle* entre la représentation des lieux dans l'hippocampe et celle du cortex préfrontal, ce qui n'est pas le cas d'autres modèles biomimétiques pour lesquels ces deux structures contiennent des redondances (Poucet et al., 2004). De nouvelles colonnes sont créées lorsqu'aucune autre colonne ne correspond suffisamment -selon un taux fixé à l'avance- au lieu occupé par l'agent. L'un des problèmes majeurs de l'implémentation choisie étant la difficulté de limiter le nombre de colonnes créées, nous avons intégré au modèle un mécanisme permettant de fusionner deux colonnes dont les lieux associés atteignent un seuil de proximité donné.

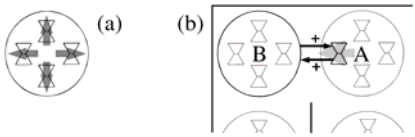


Fig. 1 – (a) *Représentation schématique* d'une hypercolonne (cercle) et des minicolonnes (en forme de sablier) représentant la direction du mouvement associé. (b) *Apprentissage des relations d'adjacence*. L'agent effectue un mouvement vers l'ouest. Des connexions réciproques sont renforcées entre la minicolonne « Ouest » de l'hypercolonne A et l'hypercolonne B.

La carte corticale correspond à un réseau de telles colonnes interconnectées où chaque connexion représente une *relation d'adjacence* entre deux lieux. L'apprentissage de cette relation concerne les mouvements à effectuer pour passer d'un lieu à un autre, mouvements qui sont codés à l'intérieur de chaque colonne (ou *hypercolonne*, Mountcastle, 1997) par 4 *minicolonnes* représentant les mouvements (Nord, Sud, Est, Ouest) pouvant être effectués depuis ce lieu. Les transitions sont apprises par apprentissage Hebbien lors de l'exploration (Fig. 1) et peuvent être oubliées si la topographie change. Chacune de ces minicolonnes est également

décomposée en deux divisions transformant le réseau d'unités en un *réseau à deux couches*, celle du haut recevant et propageant des informations *motivationnelles liées au but recherché* et celle du bas des informations *sensorielles liées à la position spatiale*. Grâce à la construction d'un tel réseau, le système peut planifier un itinéraire complet permettant d'atteindre un but (Fig. 2).

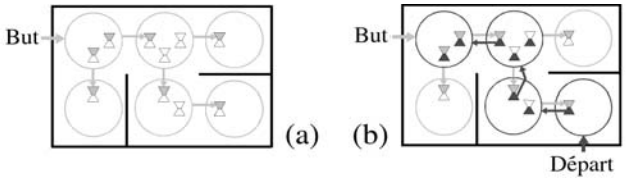


Fig 2. – (a) Le *réseau de recherche* planifie les séquences motrices pour atteindre le but. (b) Le *réseau d'exécution* indique la suite d'action à effectuer, ici « Ouest », « Nord », « Ouest ».

Tout d'abord un *réseau de recherche* est élaboré en rétropropageant un signal depuis l'hypercolonne associée à la position d'un but via les connexions reliant toutes les divisions hautes de ses minicolonnes à la division haute d'une des minicolonnes des hypercolonnes adjacentes (et ainsi de suite). Une fois que le signal atteint l'hypercolonne associée à la position courante de l'agent, un *réseau d'exécution* propage un autre signal via les connexions provenant des divisions basses des minicolonnes précédemment activées (connexions basses réciproques à celles précédemment décrites), ce qui établit la séquence de mouvements reliant la position courante à celle du but. Le plus court chemin sera défini par la vitesse de propagation de ce signal.

Ce modèle a été testé dans le cadre d'une tâche de planification classique qui propose trois chemins de longueur croissante permettant d'atteindre un même but (Tolman & Honzik, 1930). Le chemin le plus court doit être utilisé tant qu'il n'est pas obstrué (Etape 1), l'agent doit ensuite emprunter le chemin de longueur intermédiaire (Etape 2) qui sera à son tour bloqué ne laissant que le chemin le plus long pour atteindre le but (Etape 3) (Fig. 3). Les tests ont été effectués sur 50 exécutions du programme et 100 fois lors d'une même exécution. Après trois explorations de l'environnement, l'agent planifie *et* exécute le bon chemin dans 96,7% des cas lors de l'étape 1, dans 95% des cas lors de l'étape 2 et dans 98,6% des cas lors de l'étape 3. Le nombre d'essais nécessaires à un changement de chemin est faible, puisqu'entre l'étape 1 et l'étape 2 il est en moyenne de 1,1 et entre l'étape 2 et l'étape 3 il est de 1,05.

3 Discussion et Conclusion

La carte topologique d'hypercolonnes corticales élaborée dynamiquement à partir d'une carte de cellules de lieux hippocampiques permet une planification rapide et efficace. Outre l'amélioration de la granularité du codage des mouvements par les

minicolonnes, la poursuite de ce travail concernera plus particulièrement l'apprentissage de la localisation des buts (ici déterminée ad hoc) par les colonnes préfrontales conforme aux données biologiques - c'est-à-dire sans référence aux récompenses associées (Hok et al., 2005). Le modèle complet constitué des deux modules hippocampique et cortical sera susceptible de proposer des stratégies différentes pour atteindre un but courant correspondant à un certain état motivationnel, stratégies qui seront sélectionnées en fonction des perceptions externes et internes par l'architecture de sélection de l'action - inspirée des circuits nerveux des ganglions de la base du rat - dont nous disposons. Ce travail vise, d'une part, à aider à la compréhension des mécanismes biologiques correspondants chez le rat et, d'autre part, à accroître l'autonomie décisionnelle du robot Psikharpax en comparant ces solutions biomimétiques à d'autres solutions « ingénieurs » (e.g., Filliat, 2001).

Références

ARLEO A., SMERALDI F. & GERSTNER W. (2004). Cognitive navigation based on nonuniform Gabor space sampling, unsupervised growing networks, and reinforcement learning. *IEEE Transactions on Neural Networks*. 15(3), p. 639-652.

BIEZCZAD A. & PAGUREK B. (1997). Running rats with Neurosolver-based brains in mazes. *Nineteenth Annual Conference of the Cognitive Science Society CogSci'97*.

DE BRUIN J.P., MOITA M.P., de BRABANDER H.M. & JOOSTEN R.N. (2001). Place and response learning of rats in a Morris water maze: differential effects of fimbria fornix and medial prefrontal cortex lesions. *Neurobiological Learning Memory*. 75(2), p. 164-178.

ETHIER K., LE MAREC N., ROMPRE P.P. & GODBOUT R. (2001). Spatial strategy elaboration in egocentric and allocentric tasks following medial prefrontal cortex lesions in the rat. *Brain Cognition*. 46(1-2), p. 134-135.

FILLIAT D. (2001). Cartographie et estimation globale de la position pour un robot mobile autonome. *Thèse de doctorat*. Université Pierre et Marie Curie - Paris VI.

GIRARD, B., FILLIAT, D., MEYER, J., BERTHOZ, A., & GUILLOT, A. (sous presse). Integration of navigation and action selection functionalities in a computational model of cortico-basal ganglia-thalamo-cortical loops. *Adaptive Behavior*.

HOK V., SAVE E., LENCK-SANTINI P.P., & POUCET B. (2005). Coding for spatial goals in the prelimbic_infralimbic area of the rat frontal cortex. *PNAS*. 102(12), p. 4602-4607.

KOENE R.A. & HASSELMO M.E. (sous presse). An integrate and fire model of prefrontal cortex neuronal activity during performance of goal-directed decision making. *Cerebral Cortex*.

MEYER, J.-A., GUILLOT, A., GIRARD, B., KHAMASSI, M., PIRIM, P., and BERTHOZ, A. (2005). The Psikharpax project: Towards building an artificial rat. *Robotics and Autonomous Systems*. 50(4), p. 211-223.

MOUNTCASTLE V.B. (1997). The columnar organization of the neocortex. *Brain*. 120, p. 701-722.

POUCET B., LENCK-SANTINI P.P., HOK V., SAVE E., BANQUET J.P., GAUSSIER P. & MULLER R.U. (2004). Spatial navigation and hippocampal place cell firing: the problem of goal encoding. *Review Neuroscience*. 15(2), p. 89-107.

REDISH A.D. (1999) *Beyond the cognitive map: From place cells to episodic memory*. MIT Press. Cambridge

TOLMAN E.C. & HONZIK C.H. (1930), Introduction and removal of reward and maze performance in rats. *University of California Publications in Psychology*. 4, p. 257-275.

Un modèle multi-agents pour l'étude des dynamiques évolutives au niveau cosmologique

Jean-Claude Torrel[1,2], Claude Lattaud[2], Jean-Claude Heudin[1]

[1] IIM Lab. - Pôle Léonard de Vinci
http://www.devinci.fr/iim/iim/
{jean-claude.heudin,jean-claude.torrel}@devinci.fr
[2] Lab. d'Intelligence Artificielle de Paris 5 - Université René Descartes
http://www.math-info.univ-paris5.fr/alife/
Claude.Lattaud@math-info.univ-paris5.fr

Mots-clés : multi-agents, complexite, modeles cosmologiques, phenomenes emergents, dynamiques evolutives.

1 Introduction

Des particules primordiales aux grandes structures, l'univers est le produit d'une évolution, celle du simple vers le complexe. Des travaux récents proposent que cette évolution soit comparable à celle que l'on peut observer au niveau biologique. Plus qu'une simple similitude, il existerait des lois générales qui expliqueraient non seulement l'évolution du monde biologique mais aussi celle du monde physique. Cette hypothèse ne contredit pas a priori le sens commun, puisque le biologique est une partie intégrante du monde physique. Nous avons effectué plusieurs travaux visant à étudier les dynamiques à différents niveaux de complexité : à bas niveau avec les automates cellulaires (Magnier *et al.*, 1997), au niveau biologique et au niveau cosmologique avec des systèmes multi-agents (Heudin, 2003). Dans cette étude nous développons un nouveau modèle multi-agents dont les caractéristiques sont les suivantes :
- Le modèle est un réseau d'agents en interactions mutuelles.
- Les agents sont les plus simples possibles.
- Les agents sont autonomes au sens où l'état d'un agent à un instant donné résulte de ses états antérieurs et de ses interactions avec les autres agents.
- Aucun agent ne contrôle directement le comportement de tous les autres agents.
- Des interactions entre agents émergent des structures (patterns) et des comportements globaux au niveau du réseau.
- En jouant sur les paramètres du modèle, on obtient un espace des instances de ce modèle couvrant large spectre des dynamiques possibles.

Notre approche est complémentaire des simulations effectuées en cosmologie. Alors que les astrophysiciens simulent l'univers tel qu'ils l'observent selon une démarche réductionniste classique strictement fondée sur les observations, nous proposons une

démarche ascendante qui permet létude des dynamiques évolutives non seulement de l'univers tel qu'on le comprends mais aussi en le replaçant dans le paysage des univers possibles.

2 Le modèle multi-agents

Le modèle est constitué d'un réseau d'agents dont les paramètres sont :
ses coordonnées $C(x, y, z)$, son rayon r, sa vitesse instantanée $V(x, y, z)$, sa masse m, sa viscosité μ, son spin $s(x, y, z)$, sa fonction d'effondrement $E()$ et une probabilité e de déclencher les fonctions :
$\rightarrow$ d'évolution du rayon $R()$: engendre une modification du rayon.
$\rightarrow$ d'évolution de la viscosité $\nu()$: fait évoluer la viscosité de l'agent.
A chaque évolution du système, tous les agents ont la possibilité (dépendante de e) de changer d'état, ce qui se traduit, lorsque cela a lieu, par une modification de leur rayon et de leur viscosité en fonction de la loi de transition interne. Ces lois d'évolution interne du rayon et de la viscosité ont pour but d'élargir les possibilités de paramétrisation du modèle, en donnant des possibilités supplémentaires à la dynamique d'évolution du système. Le paramètre qui détermine la viscosité du gaz est un pourcentage: le spin et le déplacement d'un agent sont influencés par l'ensemble des agents locaux, c'est-à-dire à une distance maximale de N (et dans des proportions qui dépendent du paramètre de viscosité μ de l'agent. Ce paramètre est un pourcentage (0% : les gaz sont des fluides non visqueux et 100% : les gaz sont considérés comme des solides.) La fonction d'effondrement agit sur le rayon des agents gazeux pour simuler l'effondrement gravitationnel des masses de gaz que nous avions noté dans les expériences préparatoires.
L'environnement est une discrétisation de l'espace sous la forme d'une grille avec un maillage de taille N, en trois dimensions. L'environnement exerce aussi une force, identique sur l'ensemble des agents, qui s'oppose à l'effondrement (donc dirigée du centre vers l'extérieur) pour simuler l'expansion de l'univers.
Ce modèle dispose de trois fonctions de transitions discrètes : F_e (une fonction inter-agents agissant sur les coordonnées), F_s (fonction inter-agents agissant sur le spin) et F_i (fonction interne aux agents). A chaque pas de temps, l'environnement applique donc les trois fonctions sur chaque agent du système.
$F_e : C_i^{t+1}(x, y, z) = C_i^t(x, y, z) + T_j^t(x, y, z) + U_j^t(x, y, z)$ où :
$C_i^t(x, y, z)$: sont les coordonnées x, y, z de l'agent i à l'instant t,
$T_i^t(x, y, z)$: l'action de la gravitation sur l'agent i à l'instant t sur les coordonnées,
$U_i^t(x, y, z)$: l'impact de la viscosité sur le déplacement de l'agent i au temps t sur les coordonnées x, y, z.
On peut détailler la fonction T ainsi en fonction des coordonnées :
$T : \frac{dx_i}{dt^2} = G \cdot \sum_{i,j=1\ /\ i \neq j}^{M} \left(\frac{m_j (x_j - x_i)}{(d_{ij}^2)^{\frac{3}{2}}} \right)$ avec :
M : le nombre d'agents présents dans l'univers,
G : la constante de gravitation,
m_i : la masse de l'agent i,
d_{ij} : la distance entre l'agent i et l'agent j.

La fonction U est l'application de la viscosité sur un agent i : $U : \delta x_i = \sum_{j=1\ si\ d_{ij} \leq N}^{M} \left[\mu_j^t \cdot \left(V_j^t(x) + s_j^t(x)\right)\right]$
avec : s_i^t : le spin de l'agent i au temps t,
μ_j^t : la viscosité de l'agent j au temps t.
La fonction F_s est définie comme suit : $F_s : s_i^{t+1}(x) = s_i^t(x) + \sum_{j=1\ si\ d_{ij} \leq N}^{M} \left(\mu_j^t \cdot s_j^t(x)\right)$
Soit e' un nombre aléatoire tiré à chaque pas de temps pour chaque agent :

$$\begin{cases} \text{si} \quad e' \leq e & \text{alors} \quad \mu_i^{t+1} = \nu(\mu_t) \quad \text{et} \quad r_i^{t+1} = R(r_i^t) \\ \text{sinon} & \mu_i^{t+1} = \mu_t \quad \text{et} \quad r_i^{t+1} = r_i^t - (p_i^t \cdot r_i^t) \end{cases}$$

avec : r_i^t : le rayon de l'agent i à l'instant t
p_i^t : le pourcentage d'effondrement de l'agent i à l'instant t

3 Validation du modèle :

Afin de valider le modèle, celui-ci doit être capable de modéliser l'observable (Alimi *et al.*, 1999). Pour cela nous allons donc étudier la dynamique d'évolution, dans le cas particulier où les paramètres correspondent à la réalité physique.

3.1. Expériences qualitatives : Nous avons comparé les résultats produits par un al-

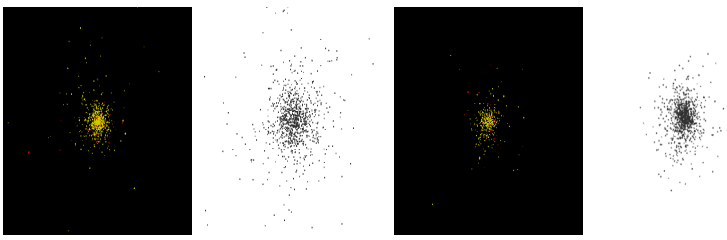


Figure 1: Evolution de la simulation pour les deux modèles à $t = 23,24$ et $t = 28,12$. Stabilisation d'un amas après une phase transitoire d'effondrement.

gorithme de type *Particle-Mesh* ([(Hocney & Eastwood, 1980)]) à ceux produits par notre modèle dans des conditions similaires. Comme nous pouvons le voir sur les images (Fig. 1) notre modèle suit la même dynamique évolutive que le *Particle-Mesh* : une phase d'effondrement au cours de laquelle les particules en rotation autour du centre raccourcissent leur orbite, une phase de stabilisation au cours de laquelle l'effondrement gravitationnel diminue jusqu'à disparaître et une phase sans fin pendant laquelle la forme globale de la structure est fixe mais son centre est agité de mouvements d'agents en rotation désordonnée.

3.2. Expériences quantitatives : Nous avons évalué la densité de matière au centre de l'univers. La courbe (Fig. 2) présente l'évolution de cette densité. En répétant l'expérience avec des conditions initiales différentes, nous avons pu remarquer que la dynamique d'évolution s'articule toujours autour des trois phases que nous avons précédement décrites, aussi bien pour le *PM* que pour notre modèle. La Fig. 2 montre

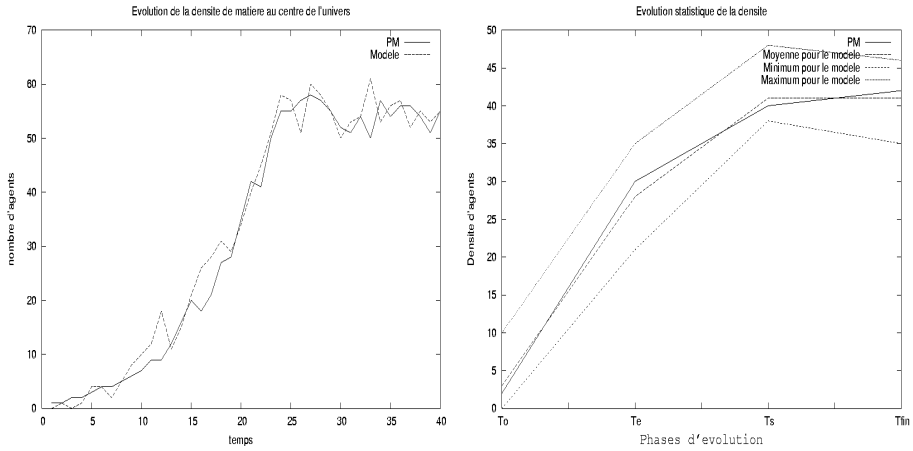


Figure 2: Statistique (sur 120 expériences) de l'évolution de densité de matière au centre de l'univers entre un PM (à gauche) et notre modèle (à droite)

la différence de densité en un point central de l'espace entre les simulations faites avec un *PM* et avec notre modèle.

4 Conclusion et perspectives

Ainsi, comme les analyses effectuées ont pu le montrer, le modèle développé est parfaitement capable de simuler l'observable, les différences notables sur les expérimentations étant dues à la différence de précision des algorithmes utilisés (cf. (Hocney & Eastwood, 1980) (Barnes & Hut, 1986)). Cette étape de validation d'un modèle multi-agents original (fonction d'évolution des agents, gestion du spin des particules et de leur viscosité) nous permet désormais de nous attacher à l'étude des classes de complexité et à leur organisation (Wolfram, 1984).

References

ALIMI J., J.P.CHIEZE, TEYSSIER R., SERMA A. & AUDIT E. (1999). Simulations numeriques en cosmologie. *Calculateurs paralleles. Vol. 11*, p. 255–273.

BARNES J. & HUT P. (1986). A hierarchical O(N log N) force-calculation algorithm. *Nature vol.324*, p. 446–449.

HEUDIN J. (2003). Complexity Classes in Three-dimensional Gravitational Agents. *Artificial Life VIII*, p. 9–13.

HOCNEY & EASTWOOD (1980). *Simulations using Particles*.

MAGNIER M., C.LATTAUD & HEUDIN J. (1997). Complexity Classes in the Two-dimensional Life Cellular Automata Subspace. *Complex Systems 11*, p. 419–436.

WOLFRAM S. (1984). Universality and Complexity in Cellular Automata. *Physica D*, p. 1–35.