Une Approche Hybride pour SAT *

Djamal Habet[1], Chu Min Li[2], Laure Devendeville[2], Michel Vasquez[1], Jean-Luc Guérin[2]

[1]Centre de recherche LGI2P
École des Mines d'Alès
Site EERIE, Parc Scientifique Georges Besse
30035, Nîmes, Cedex 01
{Djamal.Habet, Michel.Vasquez}@ema.fr

[2]LaRIA
Univ. de Picardie Jules Vernes
5, Rue du Moulin Neuf
80000, Amiens
{cli, devendev, guerin}@laria.u-picardie.fr

Résumé

On observe, dans les problèmes SAT, des dépendances structurelles entre les variables. Leur exploitation par la recherche locale a montré son efficacité [1]. Dans cet article, une extension de cette approche est proposée par une hybridation d'un algorithme de recherche locale (Walksat[13, 15]) et de la procédure de Davis-Putnam (Satz[10]). A chaque nœud de l'arbre de recherche construit par Satz, jusqu'à une profondeur fixée, nous construisons un graphe d'implications entre les littéraux. Ses composantes fortement connexes correspondent aux classes d'équivalences. Ce graphe est donc réduit par substitution des littéraux par leur unique représentant de classe. Enfin, la recherche de la fermeture transitive des implications est réalisée. Ces relations structurelles seront exploitées par Walksat à chacun des nœuds de l'arbre de recherche de Satz. Ce procédé est fortement motivé par la puissance de l'heuristique de branchement de Satz. En effet, elle peut fournir un chemin valide vers la solution ainsi qu'un nombre croissant d'implications aux nœuds les plus profonds. Les résultats expérimentaux montrent l'efficacité de notre approche.

Mots-clé : Procédure Davis-Putnam-Loveland, recherche locale, relations structurelles.

1 Introduction

Considérons une formule propositionnelle F, sous forme normale conjonctive, définie sur un ensemble de variables booléennes $\{x_1, x_2, \dots, x_n\}$. Le problème de Satisfiabilité (SAT) consiste à répondre sur l'existence d'une affectation booléenne à ces variables satisfaisant toutes les clauses de F.

SAT est fondamental dans de nombreux domaines de l'informatique, ingénierie électrique, mathématiques, etc. Il fût le premier problème défini NP-complet [2].

*Ce travail a été partiellement financé par le CNRS sous la subvention N° SUB/2001/0111/DR16

L'état de l'art pour SAT offre deux catégories d'algorithmes de résolution. Les méthodes stochastiques, basées sur la recherche locale, aboutissent rapidement à une solution mais se heurtent aux minima locaux de leurs fonctions objectives et n'exploitent guère la structure de la formule SAT. D'autre part, les méthodes complètes, basées sur la méthode de Davis-Putnam-Loveland (DPL) [4], dépendent fortement du choix de la variable de branchement. Toutefois, Satz [9, 10] offre une puissante heuristique de branchement aboutissant à des formules plus réduites et fournissant un bon chemin vers la solution.

L'exploitation des relations structurelles entre les variables lors de la recherche locale est l'un des dix challenges proposés par Selman *et al.* [14]. Dans l'algorithme TSAT [1], Devendeville, Saïs et Grégoire utilisent un prétraitement basé sur la propagation unitaire pour extraire ces relations et les intégrer au mécanisme de réparation dans une recherche locale.

Dans cet article, nous présentons une exploitation plus approfondie des relations structurelles à travers une hybridation de Walksat [13, 15] et Satz. Nous étendons ces relations aux équivalences par la recherche des composantes fortement connexes du graphe des implications. Chaque classe est signifiée par un représentant substituant les éléments restants afin de simplifier la formule initiale. Cette opération est suivie par la construction de la fermeture transitive des implications entre les représentants des classes. A ce stade, Walksat sera appliqué en exploitant ces différentes constructions. Ce procédé est répété à chaque nœud nouvellement atteint par Satz limité à une profondeur maximale.

Précédemment, Mazure, Saïs et Grégoire [12] ont intégré la recherche locale à chaque nœud d'un arbre de recherche DPL dans le but d'aider le choix de la variable de branchement. L'approche que nous proposons dans ce papier est inverse: le branchement de Satz est utilisé afin d'aider la recherche locale.

Après une présentation des deux algorithmes hybridés, Walksat en section 2 et Satz en section 3, nous détaillons notre approche en section 4. Les résultats exposés en section 5 sont discutés en section 6. Enfin, la section 7 conclut.

2 Walksat

Les méthodes de recherche locale effectuent un parcours non systématique de l'espace de recherche. Walksat vise à minimiser le nombre total des clauses non satisfaites en effectuant une descente graduelle dans l'espace de recherche [15]. Il aboutit éventuellement à une affectation satisfaisant la formule SAT.

Walksat ne fût défini que dans [13]. Dans une boucle *essai*, Walksat tente de corriger une affectation initiale, générée aléatoirement, par un nombre maximal de réparations en inversant la valeur d'une variable intervenant dans les clauses violées. Ces essais sont réitérés jusqu'à ce qu'un maximum soit atteint ou une solution soit trouvée. La fonction *score(x)* calcule, pour la variable x, le nombre total de clauses, impliquants x, violées par une instanciation générée par une éventuelle réparation de x. Cette fonction est utilisée pour choisir la variable à réparer. Ceci donne l'algorithme Walksat suivant:

Procédure Walksat
Entrée: Une formule SAT F, MAX-ESSAIS, MAX-RÉPARATIONS
Sortie: Une affectation T satisfaisant F, si trouvée
Début
  Pour essai = 1 **à** MAX-ESSAIS
    $T \leftarrow$ Affectation aléatoire des variables de F;
    Pour réparation = 1 **à** MAX-RÉPARATIONS
      Si T satisfait F retourner T;
      $c \leftarrow$ Une clause violée choisie aléatoirement;
      Calculer le score de chaque variable de c
        donnant le nombre de clauses violées si réparation;
      Si il existe des variables à score nul $v \leftarrow$ en choisir une aléatoirement;
      Sinon
        Avec une probabilité P: $v \leftarrow$ la variable à score minimal;
        Sinon $v \leftarrow$ une variable aléatoirement sélectionnée de c;
      $T \leftarrow T$ avec v réparée;
    Fin pour
  Fin pour
  retourner "Solution non trouvée";
Fin

Dans le pire des cas, le choix de la variable à réparer correspond à une marche aléatoire. Cette marche est effectuée lorsqu'aucune variable n'a de score nul. De ce fait, Walksat, en opérant sur un voisinage dynamique et restreint, obtient des meilleures résultats que ces prédécesseurs [7].

3 Satz

Introduite en 1960 et en dépit de sa simplicité la procédure de Davis-Putnam-Loveland (DPL) [4] subsiste l'une des meilleures méthodes complètes pour SAT. La procédure DPL construit un arbre de résolution binaire à travers l'espace de recherche et chaque appel récursif constitue un nouveau nœud de cet arbre. Toutes les feuilles, exceptées éventuellement une pour les formules satisfiables, représentent l'aboutissement de la procédure à une contradiction.

Procédure DPL
Entrée: Une formule SAT F
Sortie: Décision SAT
Début
  Si F est vide retourner "Satisfiable";
  $F \leftarrow$ PropagationUnitaire(F);
  Si F contient une clause vide retourner "Non satisfiable";
  /* Règle de branchement */
  Sélectionner une variable x dans F selon une heuristique H;
  Si DPL (F$\cup\{x\}$) retourne "Satisfiable"
    retourner "Satisfiable";
  Sinon
    retourner DPL (F$\cup\{\bar{x}\}$)
Fin

Procédure PropagationUnitaire
Entrée: Une formule SAT F
Sortie: F simplifiée par propagations unitaires
Début
TantQue Pas de clauses vides et il existe une clause unitaire l dans F
Affecter 1 à l et simplifier F;
Fin TantQue
retourner F;
Fin

Le choix de la variable de branchement est un facteur primordial de la procédure DPL. Plusieurs heuristiques ont été conçues. L'heuristique MOM[1] [6], d'une efficacité et d'une simplicité étonnantes, consiste à se brancher à la variable ayant le maximum d'occurrences dans les clauses les plus courtes. Néanmoins, les travaux réalisés dans Posit [5, 6], Tableau [3] et Satx [9] suggèrent d'intégrer la propagation des clauses unitaires à cette heuristique. Cela s'effectue par l'ajout respectif des clauses unitaires x et $\overline{x}$ dont l'effet de leurs propagations pondère la variable x. Une telle heuristique, dite UP[2], permet une prise en compte efficace de toutes les clauses unitaires et la détection des littéraux échoués.

Cependant, à cause d'un coût de traitement élevé, il est nécessaire de réduire le nombre de variables à examiner par l'heuristique UP. Satz aboutit à l'une des meilleures restrictions par la prise en compte du nombre d'occurrences binaires des variables candidates au branchement. Ainsi, Satz tire avantage des techniques issues de la propagation unitaire combinées à l'heuristique MOM en réduisant la taille de l'arbre de recherche, et en anticipant l'aboutissement à des contradictions sur des chemins non valides.

4 Approche hybride

Le principe de notre approche est d'exploiter conjointement les avantages de Walksat et Satz. En effet, la règle de branchement utilisée par Satz trouve un bon chemin vers une solution, si elle existe. Ce chemin peut être utilisé avantageusement par Walksat tout en respectant des dépendances non formulées explicitement par la formule SAT initiale.

4.1 Dépendances entre variables

L'étude approfondie de nombreuses classes de problèmes montre l'existence de dépendances sous forme d'implications et d'équivalences entre variables. La notion de dépendance (affaiblissement de la notion proposée dans [8]) est définie par:

Définition: dépendance
Soient Σ un ensemble de clauses, V l'ensemble des variables apparaissant dans Σ, C une conjonction finie de littéraux dont les variables $V_C \subseteq V$ et y une variable de V telle que $\Sigma \models (y \Rightarrow C)$. On dira alors que l'ensemble V_C dépend de la variable y. Plus simplement,

1. Maximum Occurrences of clauses of Minimum size
2. Unit Propagation

si l'on fixe y alors les variables de V_C seront fixées elles-aussi [4].

Exemple: soit l'ensemble des clauses $\Sigma = \{\neg l \vee a,\ \neg l \vee \neg b,\ b \vee \neg c\}$. Fixer l fixera les littéraux a, $\neg b$, $\neg c$ et donc la construction des dépendances $l \rightarrow \{a, \neg b, \neg c\}$ [4].
Une dépendance ainsi décrite est une relation d'implication.

La construction de la liste des littéraux dépendants se fait par simples propagations unitaires. Cette opération est naturellement effectuée par Satz, lors de la recherche de la variable de branchement, en un temps linéaire. La construction complète des dépendances est représentée par un graphe orienté des implications où les nœuds sont des littéraux et les arcs relient deux littéraux dépendants.

La réflexivité de l'implication entre deux littéraux signifie leur équivalence. Cette relation d'équivalence est une dépendance plus forte et exprimée par les composantes fortement connexes du graphe des implications. Leur construction aboutit à un graphe d'implications réduit et acyclique. Et la formule SAT se voit aussi réduite par le regroupement des littéraux équivalents en un seul littéral représentant.

4.2 Consistance des dépendances

Lors de ses prétraitements et à chaque branchement, Satz marque certaines variables à l'état *passif* indiquant leur instanciation. Cependant et afin de garder la cohérence des implications construites, cet état doit être propagé à travers le graphe des implications pour les littéraux satisfaits et à travers son graphe transposé pour les littéraux falsifiés. Ces propagations s'opèrent, en un temps linéaire, par des parcours en profondeur sur les graphes respectifs. Ces deux parcours permettent aussi la construction des classes d'équivalence en ne considérant que les variables *actives*.

Par ailleurs, la relation d'implication étant transitive, sa complétude doit être assurée par la construction de sa fermeture transitive sur l'ensemble des implications entre les classes d'équivalence. Ce qui s'opère, encore une fois, par un parcours en profondeur sur le graphe réduit des implications. Ainsi, l'ensemble des implications est complet et consistant.

4.3 Algorithme proposé: *Walksatz*

La formule SAT résultante, à chaque nœud de l'arbre de recherche de Satz à profondeur maximale fixée, fera l'objet des traitements énumérés précédemment. A ce stade, Walksat traite une formule réduite à ses classes d'équivalence où chacune est représentée par un unique littéral choisi arbitrairement. Le procédé de sélection de la classe à réparer reste identique à ce qui a été décrit en section 2.

La réduction d'un problème par le regroupement des littéraux équivalents peut engendrer une diminution sensible du nombre réel des variables à traiter. Cependant, cela occasionne, dans des cas indésirables, l'annulation d'actions successives sur la même entité et une perte inutile de réparations. Ce phénomène est corrigé par une liste tabou [11] de longueur fixée empiriquement à 1.

Walksat fournit trois niveaux de sélection de variable à réparer. Pour garantir une bonne intégration des implications dans différentes combinaisons de ces niveaux, une étude empirique a été effectuée. Un comparatif des résultats montre que les meilleures performances s'observent par adjonction des implications à satisfaire lors de la minimisation du nombre total des clauses violées. L'interprétation de ce constat repose sur :

1. la réparation de la variable à score nul fait progresser, le mieux, vers une solution globale. Mais la réparation des classes impliquées n'assure pas le maintien de cette progression;
2. la marche aléatoire introduite dans Walksat est incompatible avec le caractère déterministe des implications. Ce dernier risque d'amplifier un aléas.

L'algorithme général, **Walksatz**, de l'ensemble des étapes décrites se résume comme suit:

Procédure *Walksatz*
Entrée: une formule SAT F
Sortie: Une affectation T satisfaisant F, si trouvée
Début
Pour Chaque nœud construit par Satz, limité à une profondeur maximale
Si F est vide retourner "Solution T, par Satz";
Construire sur F le graphe des implications I et son transposé I^t;
Propager l'état des variables sur I et I^t;
Construire les classes d'équivalence et réduire F, I et I^t;
Construire la fermeture transitive des implications sur I réduit;
Appliquer Walksat à F réduite avec prise en charge de I et I^t réduits lors de la minimisation du nombre de clauses violées;
Si la dernière étape retourne "Solution Trouvée T"
retourner "Solution T, par Walksat";
Fin pour
retourner "Solution non trouvée;
Fin

5 Résultats expérimentaux

L'approche hybride est une combinaison d'une méthode exacte et d'une méthode incomplète. De ce fait, il est difficile d'annoncer des critères comparatifs aux techniques introduites dans ces deux familles. Cependant, pour analyser le comportement de Walksatz par rapport à Walksat et Satz, nous avons eu recours à un ensemble de benchmarks [3] utilisés dans l'évaluation de toutes les études SAT.

Walksatz est codé sous Linux/C++ et compilé avec g++ sur une machine Duron 800 Mhz avec 256 MO de mémoire vive. Les paramètres retenus dans nos expérimentations sont:

	Walksatz	Walksat
MAX-ESSAIS	1	100
MAX-RÉPARATIONS	10^5	15×10^5
Nombre de Lancements	100	1

3. Disponibles sur le site Internet www.intellektik.informatik.tu-darmstadt.de/SATLIB/Benchmarks

Il est à noter que la profondeur de l'arbre de Satz est limitée à 3. La racine étant à la profondeur 0, quinze noeuds sont développés. Le nombre total de réparations de Walksatz est donc identique à celui de Walksat. Les colonnes T%, # rép. et sec., correspondent respectivement au taux de réussite, au nombre moyen des réparations et au temps moyen de calcul en secondes. Les valeurs de # rép. et sec. sont calculées sur les lancements réussis.

5.1 Les benchmarks DIMACS

La classe "aim": définies par Iwama *et al.*, sont des instances 3-SAT difficiles. Chaque instance n'a qu'une seule solution.

AIM			Walksatz			Walksat			Satz
Instance	# Vers.	# Cls.	T%	# rép.	sec.	T%	# rép.	sec.	sec.
aim-50-1-6-yes1-1	50	300	100	1	0.001	6	920684	44.422	0.020
aim-50-1-6-yes1-2	50	300	100	1	0.000	1	64016	148.600	0.030
aim-50-1-6-yes1-3	50	300	100	1	0.000	34	685077	6.378	0.010
aim-50-1-6-yes1-4	50	300	100	1	0.000	0	-	-	0.010
aim-50-2-0-yes1-1	50	100	100	1	0.000	2	702	83.125	0.010
aim-50-2-0-yes1-2	50	100	100	1	0.000	97	22170	0.094	0.020
aim-50-2-0-yes1-3	50	100	100	1	0.000	100	151431	0.287	0.020
aim-50-2-0-yes1-4	50	100	100	1	0.000	100	137124	0.237	0.020
aim-100-1-6-yes1-1	100	160	100	1	0.000	0	-	-	0.020
aim-100-1-6-yes1-2	100	160	100	1	0.000	0	-	-	0.030
aim-100-1-6-yes1-3	100	160	100	1	0.000	0	-	-	0.030
aim-100-1-6-yes1-4	100	160	100	1	0.000	0	-	-	0.030
aim-100-2-0-yes1-1	100	200	100	1	0.000	0	-	-	0.020
aim-100-2-0-yes1-2	100	200	100	1	0.001	0	-	-	0.030
aim-100-2-0-yes1-3	100	200	100	1	0.000	0	-	-	0.030
aim-100-2-0-yes1-4	100	200	100	1	0.000	0	-	-	0.020
aim-200-1-6-yes1-1	200	360	100	1	0.003	0	-	-	0.050
aim-200-1-6-yes1-2	200	360	100	1	0.003	0	-	-	0.090
aim-200-1-6-yes1-3	200	360	100	1	0.000	0	-	-	0.050
aim-200-1-6-yes1-4	200	360	100	1	0.001	0	-	-	0.050
aim-200-2-0-yes1-1	200	400	100	1	0.001	0	-	-	0.050
aim-200-2-0-yes1-2	200	400	100	1	0.001	0	-	-	0.040
aim-200-2-0-yes1-3	200	400	100	1	0.001	0	-	-	0.100
aim-200-2-0-yes1-4	200	400	100	1	0.001	0	-	-	0.070

TAB. 1 – *Résultats obtenus sur les instances "aim"*

La classe "ssa": ces instances sont définies par A. V. Gelder *et al.* et correspondent au problème de *single-stuck-at-fault* de l'analyse des circuits.

SSA			Walksatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# rép.	sec.	T%	# rép.	sec.	sec.
ssa7552-160	1391	3126	100	385	2.270	100	30875	0.061	0.080
ssa7552-159	1363	3032	100	344	2.265	100	22006	0.433	0.080
ssa7552-158	1363	3034	100	255	2.218	100	26738	0.052	0.070
ssa7552-038	1501	3575	100	991	2.581	100	80673	0.167	0.120

TAB. 2 – *Résultats obtenus sur les instances "ssa"*

La classe "par8": issues d'un problème d'apprentissage automatique pour la fonction de parité, ces instances ont été définies par J. Crawford.

PAR			Walksatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# rép.	sec.	T%	# rép.	sec.	sec.
par8-1-c	64	254	100	12062	0.024	100	10628	0.025	0.02
par8-1	350	1149	100	68175	0.180	4	977367	79.366	0.01
par8-2-c	68	270	100	1	0.003	100	15504	0.036	0.02
par8-2	350	1157	100	120942	0.356	2	483886	160.791	0.02
par8-3-c	75	298	100	35293	0.069	100	37213	0.089	0.02
par8-3	350	1171	100	96110	0.272	3	538312	107.811	0.02
par8-4-c	67	266	100	1	0.000	100	41423	0.098	0.02
par8-4	350	1155	100	22966	0.681	0	-	-	0.02
par8-5-c	75	298	100	12843	0.027	100	34534	0.084	0.02
par8-5	350	1171	100	77604	0.284	1	304169	330.066	0.03

TAB. 3 – *Résultats obtenus sur les instances "par8"*

5.2 Vérification des microprocesseurs multiscalaires

Ces instances sont définies par M.N. Velev. Elles sont issues de la vérification des exceptions, des unités fonctionnelles multicycles et de la prédiction des branchements dans les microprocesseurs multiscalaires DLX. Dans le tableau ci-dessous, la mention "$\succ$ 7200" signifie que Satz nécessite plus de 7200 secondes pour aboutir à une solution.

DLX2_CC			Walksatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# rép.	sec.	T%	# rép.	sec.	sec.
dlx2_cc_bug01	1515	12808	100	989391	49.607	79	603523	24.857	$\succ$ 7200
dlx2_cc_bug02	1515	12808	100	858251	43.818	80	517174	21.958	$\succ$ 7200
dlx2_cc_bug03	1515	12808	100	1013661	51.013	71	723533	33.193	$\succ$ 7200
dlx2_cc_bug14	1516	12811	100	458607	26.524	98	381390	10.122	$\succ$ 7200
dlx2_cc_bug16	1516	12811	100	461816	25.555	92	417319	13.432	$\succ$ 7200
dlx2_cc_bug33	1516	12798	100	636107	23.721	99	265186	6.820	$\succ$ 7200
dlx2_cc_bug34	1516	12718	100	965240	43.048	98	369863	9.282	$\succ$ 7200
dlx2_cc_bug38	1515	12783	100	652728	35.267	63	516764	1.404	$\succ$ 7200
dlx2_cc_bug39	1482	12111	93	1141143	49.067	66	640518	33.861	$\succ$ 7200
dlx2_cc_bug40	1520	12811	100	629230	33.210	98	358702	9.688	$\succ$ 7200

TAB. 4 – *Résultats obtenus sur les instances "dlx_cc"*

5.3 Carré latin

Un "carré latin" indexé par S est un tableau $\mid S \mid \times \mid S \mid$ tel que chaque colonne et chaque ligne est une permutation des éléments de S. Ces instances sont définies par H. Zhang.

Carré latin			Walksatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# rép.	sec.	T%	# rép.	sec.	sec.
qg1-08	512	148957	6	1307271	1499.446	0	-	-	125.220
qg2-07	343	68083	100	39719	0.901	25	693660	694.622	24.340
qg3-08	512	148957	100	238622	1.505	9	628604	148.794	0.210
qg4-09	512	10469	83	701706	9.169	1	328792	3226.650	0.750
qg5-11	1331	64054	8	277167	370.769	0	-	-	4.830
qg6-09	729	21844	100	368	0.664	0	-	-	0.830
qg7-09	729	22060	100	81	0.824	0	-	-	0.970

TAB. 5 – *Résultats obtenus sur les instances "qg"*

5.4 Les benchmarks Bejing-Challenge

Ces benchmarks sont les instances SAT utilisées pendant *l'International Competition and Symposium on Satisfiability Testing* à Pékin-Chine en 1996. Ils incluent des instances de planification et d'ordonnancement.

Bejing			Walksatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# rép.	sec.	T%	# rép.	sec.	sec.
3blocks	370	13732	100	26768	0.222	100	49960	0.626	0.060
4blocks	900	59285	34	519608	102.429	0	-	-	0.060
4blocksb	540	34199	100	264532	3.655	19	694517	210.913	0.650
e0ddr2-10-by-5-1	19500	108887	100	246496	1067.484	0	-	-	137.820
e0ddr2-10-by-5-4	19500	104527	98	463371	3116.833	18	1202849	132.465	185.900
enddr2-10-by-5-1	20700	111567	97	292415	2352.466	0	-	-	112.960
enddr2-10-by-5-8	21000	113729	100	261973	1463.593	2	1329010	1448.225	105.230
ewddr2-10-by-5-1	21800	118607	97	252841	1119.552	0	-	-	401.460
ewddr2-10-by-5-8	22500	123329	100	205542	846.741	1	1456023	3136.267	120.640

TAB. 6 – *Résultats obtenus sur les instances Bejing*

5.5 Les problèmes de planification de Kautz & Selman

Proposées par H. Kautz et B. Selman, ces instances correspondent au problème de planification.

Planification			Walksatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# rép.	sec.	T%	# rép.	sec.	sec.
logistics.a	828	6718	100	21508	0.088	100	133275	0.469	2.750
logistics.b	843	7301	100	36783	0.142	100	249287	0.887	0.090
logistics.c	1141	10719	100	77653	0.334	87	490317	2.896	0.420
logistics.d	4713	21991	100	286041	155.530	74	624500	6.990	507.490
bw_large.a	459	4675	100	5146	0.199	100	14359	0.057	0.070
bw_large.b	1087	13772	100	96838	1.899	86	564590	6.249	0.240
bw_large.c	3016	50157	100	198672	22.738	0	-	-	2.740

TAB. 7 – *Résultats obtenus sur les instances de planification de Kautz & Selman*

6 Discussion

Globalement, Walksatz réduit sensiblement le nombre de réparations nécessaires pour aboutir à une solution et a un meilleur comportement face à des instances relativement difficiles pour Walksat. Les arguments suivants justifient ces constations:

1. le regroupement des littéraux équivalents permet de réduire la taille du problème;
2. la fermeture transitive des implications sur les classes d'équivalence constitue, d'une certaine manière, un tabou permettant de restreindre le voisinage lors d'une recherche locale. De plus, assurer la consistance des implications, permet de vérifier des contraintes non explicites dans la formule de départ, d'anticiper leur satisfaction et d'économiser ainsi des réparations;
3. la propagation unitaire et le choix judicieux de la variable de branchement effectués par Satz permettent de réduire la taille de la formule initiale. En effet, les formules résultantes à deux nœuds fils dans l'arbre de résolution de Satz, sont disjointes. Cela diversifie l'espace de recherche exploré par Walksat. Ce qui est constaté sur les instances DIMACS de type “aim”.

D'autre part, la construction des classes d'équivalence élimine les implications cycliques. La recherche de la fermeture transitive de ces implications permet de bénéficier totalement de l'apport de cette dépendance. Cet apport est bien illustré par la comparaison avec les résultats obtenus par TSAT qui n'intègre que partiellement le calcul des implications et qui ne tient pas compte des équivalences. De plus, la résolution par Walksatz des instances “dlx2_cc”, difficiles pour Satz, conforte la compilation de Walksat à Satz et les apports mutuels de ces deux algorithmes.

La supériorité du temps de calcul consommé par Walksatz sur certaines instances est due aux parcours en profondeur nécessaires à l'ensemble des traitements. Cela peut être facilement amélioré par une construction totalement incrémentale des classes d'équivalence et des implications lors des branchements et des *backtracks* effectués par Satz.

En dépit de ces observations, les résultats obtenus montrent l'apport de notre approche qui améliore significativement les performances de Walksat. Enfin, le respect des dépendances structurelles entre les variables est un facteur déterminant pour Walksatz.

7 Conclusion

L'étude des méthodes exactes et incomplètes appliquées à SAT, nous a conduit à trouver un compromis, sous la forme d'une approche hybride, afin d'utiliser conjointement leurs avantages respectifs. Deux méthodes, Satz et Walksat, ont participé à la concrétisation de cette approche. En maximisant l'effet de la propagation unitaire, Satz trace, dans l'espace des solutions, un arbre binaire optimal. Par ailleurs, Walksat offre un mécanisme de recherche locale relativement efficace. D'autre part, les problèmes SAT présentent des dépendances structurelles entre les variables. Ces dépendances ne sont pas exploitées par la recherche locale alors que, manifestement, elles influent sur le comportement des algorithmes SAT et réduisent la taille du problème initial.

Une combinaison bien élaborée de ces trois éléments est mise en œuvre dans Walksatz. Elle est le résultat d'une multitude de tests effectués afin d'aboutir à la meilleure imbrication des implications et des équivalences. L'expérimentation de Walksatz sur des problèmes SAT issus de divers domaines de la littérature confirme la pertinence de cette approche.

En perspective de ces travaux, l'hybridation de Satz avec des variantes de Walksat, telles que Novelty et R-Novelty [13], sont à considérer avec intérêt. D'autre part, les dépendances évoquées doivent faire l'objet d'une étude plus fine afin d'augmenter leur apport tout en optimisant le code de leur implémentation machine, par le biais d'une construction totalement incrémentale.

Références

[1] L. Brisoux Devendeville, L. Saïs, and E. Grégoire. Recherche locale: Vers une exploitation des propriétés structurelles. In *Actes de JNPC'2000*, pages 243–244, Marseille, France, 2000.

[2] S. Cook. The Complexity of Theorm Proving Procedures. In *Proceeding of the Third Annual ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[3] J.M. Crawford and L.D. Auton. Experimental Results on the Croosover Point in Random 3-SAT. In *Artificial Intelligence*, pages 151–158, Ohio, 1971.

[4] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. In *Communication of ACM*, pages 5(7):394–397, July 1962.

[5] J.W. Freeman. *Improvements to Propostional Satisfiabilty Search Algorithms.* PhD thesis, Departement of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1995.

[6] J.W. Freeman. Hard Random 3-SAT Problems and the Davis-Putnam Procedure. In *Artificial Intelligence*, pages 81:183–198, 1996.

[7] H.H. Hoos and T. Stutzle. Characterzing the run-time Behaviour of Stochastic Local Search. Technical report, AIDA-98-1, FG Intellektik, TU Darmstad, January 1998.

[8] J. Lang and P. Marquis. Complexity Results for Independance and Definability in Propositional Logic. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representataion and Reasoning (KR'98)*, pages 356–367, 1998.

[9] C. M. Li. Exploiting Yet More the Power of Unit Clause Propagation to solve 3-SAT Problem. In *ECAI'96 Workshop on Advances in Propositional Deduction*, pages 11–16, Budapest, Hungray, 1996.

[10] C. M. Li. and Anbulagan. Heuristic Based on Unit Propagation for Satisfiability. In *Proceedings de CP-97, Springer-Verlag, LNCS 1330*, pages 342–356, Austria, 1997.

[11] B. Mazure, L. Sais, and E. Grégoire. Tabu Search for SAT. In *Proceedings of AAAI'97*, pages 281–285, 1997.

[12] B. Mazure, L. Sais, and E. Grégoire. Boosting Complete Techniques Thanks to Local Search. *Annals of Mathematics and Artificial Intelligence*, 22:309–322, 1998.

[13] D. McAllester, B. Selman, and H. Kautz. Evidence for Invariants in Local Search. In *Proceedings of AAAI'97*, pages 321–326. MIT Press, 1997.

[14] B. Selman, H.Kautz, and D. McAllester. Ten Challenges in Propositional Reasoning and Search. In *Proceedings of IJCAI-97*, pages 50–54, Nagoya, Aichi, Japan, August 1997.

[15] B. Selman, H. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In MIT press, editor, *Proceedings of AAAI'94*, pages 46–51, 1994.