Académie de Montpellier
– Université Montpellier II –
Sciences et Techniques du Languedoc

Mémoire de stage de Master 2

Recherche en Informatique

Clustering de motifs séquentiels

Application à la détection d'intrusions

Hassan Saneifar

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM)

Sous la direction de :
Sandra Bringay
Anne Laurent

· Juillet 2008 ·

Remerciements

Je tiens à remercier en tout premier lieu mes deux directrices Sandra Bringay et Anne Laurent qui ont dirigé ce travail de recherche de Master. Je les remercie pour tout ce qu'elles m'ont apporté, pour leurs conseils, leur présence, leur patience, pour m'avoir fait confiance et m'avoir laissé la liberté nécessaire à l'accomplissement de mes travaux, tout en y gardant un œil critique et avisé. Je les remercie également pour plusieurs relectures de ce rapport.
Je remercie également Maguelonne Teisseire, la responsable de l'équipe TATOO du LIRMM pour m'avoir accepté au sein de l'équipe et pour le dynamisme de l'équipe. C'est un grand plaisir de travailler dans une équipe aussi active. Je remercie aussi Mathieu Roche, pour tout ce qu'il m'a apporté pendant le Master I et aussi le Master II surtout sur le domaine de fouille de textes. J'adresse également mes remerciements à Pascal Poncelet pour tout ce qu'il m'a apporté sur l'extraction de connaissances et ses conseils.

J'exprime ma reconnaissance à M. Florent Masseglia pour m'avoir donné ses conseils et ses avis sur ce projet en m'accueillant à l'INRIA Sophia Antipolis. Je le suis aussi reconnaissant pour le financement de ce projet au travers l'INRIA.

Mes remerciements vont également à Mme. Isabelle Mougenot et M. Jean-François Baget pour avoir accepté être les rapporteurs de ce travail. Je remercie beaucoup M. Rodolphe Giroudeau, responsable de Master II informatique à l'université montpellier II. J'adresse mes reconnaissances à tous mes professeurs pendant mes édudes notamment ceux du Master.

Merci à Lisa Di-Jorio pour le temps qu'elle a passé avec moi pour discuter mes propositions et pour ses aides. Merci à Chedy Raïssi pour l'intérêt qu'il a porté à mon travail et ses conseils

sur ma proposition. Je remercie en même temps de Marc Plantevit, Céline Fiot et les autres membres de l'équipe pour leurs conseils occasionnels et leurs aides. Je les remercie tous aussi de former une équipe dynamique et très accueillante.

Finalement, un grand Merci chaleureux et de tout mon cœur à mes parents "Mohammad" et "Effat", sans qui je ne serais absolument pas où j'en suis aujourd'hui. Je les remercie sincèrement pour leur gentillesse et leur soutien inconditionnel et constant, pour m'avoir donné du courage et de l'espoir, pour être toujours présents même à distance. Je leur dois ce que je suis. Aussi un Merci de tout mon coeur à mon cher frère "Mahdi" et ma chère soeur "Roghaieh" pour m'avoir donné l'occasion d'avoir deux amis véritables dans ma vie, pour leur gentillesse et leur encouragement à continuer mes études. Je vous remercie de tout mon cœur.

Résumé Les systèmes de détection d'intrusion (IDS) est aujourd'hui un composant indispensable dans le structure de sécurité de réseau. Les logs des connexions et des activités de réseaux, ayant une grande quantité d'informations, peuvent être utilisé afin de détecter les intrusions. Pour traitements ces logs, les techniques de l'extraction de connaissances (ECD) sont bien efficaces. ECD nous permet d'extraire les connaissances existant sur ces logs et les utiliser pour identifier les intrusions. La détection d'anomalie est une technique de détection d'intrusion. Il s'agit d'identifier les comportements non généraux. Pour cela, il faut modéliser les comportements et des activités généraux (normaux) de réseau. Les déviations par rapport à ces comportements sont identifiées comme des intrusions. L'efficacité de cette technique dépend de la précision et la flexibilité de la modélisation de comportements normaux.
Nous présentons une approche de la détection d'anomalies fondée sur le clustering de motifs séquentiels qui nous permet de modéliser les comportements étant comme des événements séquentiels et fréquents au fils du temps. Pour création des profils de comportements, nous regroupons les motifs séquentiels similaires par une technique de clustering adapté au motifs séquentiels.
Pour réaliser ce clustering, on a besoin d'une mesure de similarité adapté aux caractéristique particulières des motifs séquentiels. Nous allons montré pourquoi les mesures existant comme Edit Distance ne sont pas pertinent pour calculer la similarité de deux motifs séquentiels. Nous définissons ensuite, une mesure de similarité pour les motifs séquentiels qui prend en compte les caractéristiques de motifs séquentiels ainsi que le sémantique. Notre mesure de similarité peut être utilisée au sein d'autres algorithmes que le clustering. En effet dans le domaine de l'extraction de motifs séquentiels sous contrainte de similarité, la compression de motifs séquentiels, la visualisation de motifs séquentiels ou lorsqu'il il a la nécessité de comparer la similarité de deux motifs séquentiels (ou séquence de données), notre mesure est une solution efficace pour calculer la similarité. Les expérimentations montrent que notre mesure de similarité se calcul très rapidement. Cela veut dire qu'elle est utilisable au sein d'autres algorithmes sans les ralentir.

Mots-Clés : Détection d'intrusions, détection d'anomalie, détection d'outlier, fouille de données, motifs séquentiels fréquents, clustering, mesure de similarité, clustering de motifs séquentiels.

Abstract The intrusion detection systems (IDS) are an essential component in the structure of network security. The connections and networking activities logs with a large amount of informations can be used to detect intrusions. For processing of these logs, Knowledge Discovery in Databases (KDD) techniques are indeed effective. EDC allows us to extract the existing knowledges on these logs and use them to identify intrusions.

The anomaly detection is a one of intrusion detection techniques that identify the general behaviours on a network. This requires normal behaviour's models. Deviations from these behaviors are identified as intrusions. The effectiveness of this technique depends on the accuracy and flexibility of modelling normal behaviour.

We present an approach to intrusion detection based on the clustering of sequential patterns. The sequential patterns allows us to model the sequential behaviors that are seen frequently time periodes which is adapted to network's activieties behaviours. To create profiles of behaviour, we regroup the similar sequential patterns by clustering technique which is adapted to the sequential patterns.

To achieve this clustering, we need a measure of similarity adapted to the special characteristics of sequential patterns. We showed why the existing measures as Edit Distance are not relevant to calculate the similarity of two sequential patterns. We then define a measure of similarity for sequential patterns that takes into account the characteristics of sequential patterns and the theirs semantics.

Our measure of similarity can be used in other algorithms that the clustering. Indeed in the extraction of sequential patterns under similarity constraint, compressing sequential patterns and sequential patterns visualisation or when one need to compare the similarity of two sequential patterns (or data sequence), our measure is an effective solution to calculate the similarity. The experiments show that our measure of similarity calculation is very fast. This means that it is usable in other algorithms without slowing them.

Keywords : Intrusion detection, anomaly detection, outlier detection, data mining, frequent sequential patterns, clustering, similarity measure, clustering of sequential patterns.

Sommaire

Table des figures

Liste des tableaux

"*Central to this type of thinking is the underlying notion of 'truth'. By means of argument which maneuvers matter into a contradictory position, something can be shown to be false. Even if something is not completely false, the garbage has to be chipped away by the skilled exercise of critical thinking in order to lay bare the contained truth.*"

Edward De Bono

Connaître, ce n'est point démonter, ni expliquer. C'est accéder à la vision.

Antoine de Saint-Exupéry

Introduction

Les organisations possèdent désormais de grandes quantités de données numérisées, sources potentielles d'informations. En effet, elles génèrent un grand nombre de logs, encore peu exploités, malgré la source importante de connaissance qu'ils représentent. Ces logs peut être utilisés pour chercher des comportements généraux des utilisateurs d'un système, les relations de causalité, les événements séquentiels, etc.
Automatiser le traitement de ces logs est une tâche importante car ils ont des caractéristiques particulières : ils sont textuels, hétérogènes, distribués, produits continuellement et surtout très volumineux ce qui rend difficile leur exploitation. Dans ce contexte, un des principaux défis est de fournir des architectures, des protocoles et des méthodes prenant en compte les caractéristiques particulières des logs.

Dans le domaine du traitement des logs de réseaux, nous allons nous intéresser à la détection d'intrusions. Une intrusion est une série d'actions qui visent à compromettre l'intégrité, la confidentialité ou la disponibilité d'une ressource. Les logs peuvent être utilisés pour détecter ces intrusions. Plusieurs domaines de recherche proposent des solutions pour traiter ces logs comme les techniques statistiques et l'Extraction de Connaissances dans les grandes bases de Données (ECD). Nous étudierons ici ces dernières. L'ECD peut être considérée comme un processus d'Extraction de Connaissances nouvelles, potentiellement utiles et ayant un degré de plausibilité, dans de grands volumes de données [Fio07].
Le processus d'extraction de connaissances se décompose en trois étapes : (1) Pré-traitement, (2) Traitement ou fouille de données (3) Post-traitement. La 1ère phase a pour but la diminution du bruit et la transformation des données sous un format adapté aux techniques de traitement. Les méthodes de fouille de données (Data mining en anglais) sont : la classification, le clustering, l'extraction de motifs séquentiels fréquents et des règles d'association. (3) Post-traitement, c'est la phase d'interprétation des connaissances extraites pour les rendre compréhensibles et utilisables par l'utilisateur final (souvent non informaticien).

Dans notre contexte, nous nous intéressons à la recherche de motifs séquentiels fréquents (Frequent sequential pattern mining) pour exploiter les logs de réseaux et détecter des intrusions. Il s'agit d'un thème de recherche activ ces dernières années. Les motifs séquentiels permettent de trouver des corrélations entre des événements séquentiels au fils du temps. Bien que ces motifs soient adaptés aux données décrivant des événements séquentiels et temporels comme les logs, on trouve peu de travaux sur la détection d'intrusions utilisant les motifs séquentiels fréquents. Nous proposons une méthode pour la détection d'intrusions qui est basée sur la modélisation des comportements habituels des utilisateurs de réseau à l'aide du clustering de motifs séquentiels.
Pour réaliser ce clustering, nous avons défini une mesure de similarité adaptée aux caractéristiques de motifs séquentiels. Cette mesure est applicable pour d'autres applications que le clustering comme l'extraction de motifs séquentiels sous contrainte de similarité [CMB02], la compression de motifs séquentiels, la visualisation de motifs séquentiels s'agissant des comportements similaires, etc.

Dans la section 1, nous allons introduire les concepts et les notions utilisés dans le rapport. Dans la section 2 nous décrivons les travaux dans le domaine de détection d'intrusions et également ceux portant sur les mesures de similarité pour les motifs séquentiels fréquents. Les propositions sont présentées dans les sections 3, 4 et 5. Dans la section 3, nous expliquons l'approche proposée pour la détection d'intrusions en modélisant les comportements des utilisateurs par motifs séquentiels. Dans la section 4, nous définissons la mesure et nous la formalisons. Notre approche de l'apprentissage de seuil dans le cadre de l'extraction de motifs séquentiels flous est expliquée dans la section 5. Les résultats obtenus lors des expérimentations sur la mesure de similarité sont détaillés dans la section 6. Enfin, nous concluons ce rapport par le bilan des apports de notre contribution ainsi que par la présentation rapide de quelques perspectives ouvertes par notre travail.

Chapitre 1

Contexte

Dans ce chapitre, nous présentons les concepts utiles à la compréhension de ce rapport. Dans un premier temps, nous présentons les principes de la détection d'intrusions. Puis, les concepts d'ECD et plus particulièrement les techniques de fouille de données comme les règles d'association, les motifs séquentiels fréquents, la classification et le clustering.

1.1 Introduction à la détection d'intrusions

Aujourd'hui, les systèmes d'informations et les réseaux informatiques occupent une place centrale dans la société moderne. Plus il y a de données enregistrées et traitées, plus il est important de sécuriser les systèmes informatiques. Une **intrusion** se définit comme une série d'actions qui tentent de compromettre l'intégrité, la confidentialité ou la disponibilité d'une ressource [SSM06].

La détection d'intrusions est le processus de suivi des événements survenus dans un système ou un réseau et l'analyse de ces événements pour trouver des signes d'intrusions. L'objectif est d'indentifier les individus utilisant le système sans avoir l'autorisation (i.e. hackers) et les individus ayant un accès légitime au système mais qui abusent de leurs privilèges (insider threat) [MHL94]. Les **systèmes de détection d'intrusions** (Intrusion Detection system (**IDS**) en anglais) peuvent être des logiciels et des matériels qui automatisent le processus d'observation et d'analyse des événements.
Pour qu'un IDS soit efficace, il doit s'executer continuellement, s'adapter aux changements de comportements et aux grandes quantités de données, être configurable, ne pas utiliser trop de ressources mémoires de la machine et après les pannes de système, être réutilisable sans nouvel apprentissage [BGFI+98].

Actuellement, ils existe deux approches principales pour la détection d'intrusions. (1) **la détection d'anomalies** (2) **la détection d'abus** (Misuse Detection en anglais). Dans

la première approche, les comportements normaux des utilisateurs du réseau sont connus et il est donc possible de construire des profils représentant ces comportements grâce à plusieurs caractéristiques comme les activités sur le réseau, etc. Une fois ces profils définis, les intrusions sont identifiées comme des déviations par rapport aux comportements normaux [SSM06], [Kum95], [SM07], [LSM98].

L'approche **Détection d'abus** (Misuse Detection) est basée sur l'identification directe des attaques. Une signature représente une type d'attaque déjà connue. La détection d'intrusions se fait par comparaison des activités de réseau avec les signatures des attaques [SSM06], [Kum95], [LSM98], [Lun90], [SM07].

L'avantage des méthodes basées sur la détection d'anomalies est leur capacité à trouver les intrusions inconnues. Toutes fois ces méthodes produissent un taux élevé de faussses alertes car toutes les déviations par rapport aux comportements généraux ne sont forcément pas des intrusions. Par exemple, un nouveau comportement normal peut être vu comme une déviation et traité comme une intrusion.

D'autre part, dans le cas de la détection d'abus, chaque instance dans le jeu de données est labelisée "normal" ou "intrusion". Un algorithme d'apprentissage est appliqué sur les données labelisées, pour que chaque intrusion soit caractérisée sous forme d'un modèle (signature d'intrusion). On identifie une nouvelle instance comme une intrusion si elle ressemble à un modèle d'intrusion. Les modèles (signature) peuvent être crées par les experts du domaine. C'est le cas dans les systèmes d'intrusions basés sur les signatures (signature-bases intrusion détection) [DEK+02]. Ces systèmes sont efficaces pour rechercher des intrusions déjà connues. Or ces systèmes ne sont pas capables de trouver les nouvelles intrusions ou les intrusions pour lesquelles les signatures n'existent pas.

1.2 Introduction au processus ECD

Le processus d'extraction de connaissances dans les bases de données (ECD), présenté sur la figure 1.1, désigne l'ensemble des opérations qui permettent d'exploiter rapidement des données stockées en grande quantités [Fio07].

Les deux premières étapes, la sélection et le prétraitement, regroupent les différentes transformations que les données doivent subir avant d'être disponibles pour l'analyse.

Une fois mises en forme, les données sont traitées par l'algorithme d'extraction, c'est l'étape de fouille de données. Cette phase consiste à utiliser des méthodes d'analyse de données et des algorithmes de recherche qui permettrent d'obtenir, dans temps acceptable, un certain nombre de schémas et de motifs caractéristiques des données [HK00]. Il s'agit d'un processus interactif d'analyse d'un grand ensemble de données, destiné à extraire des connaissances exploitables.

Après la phase de fouille de données, les connaissances extraites ne sont pas directement

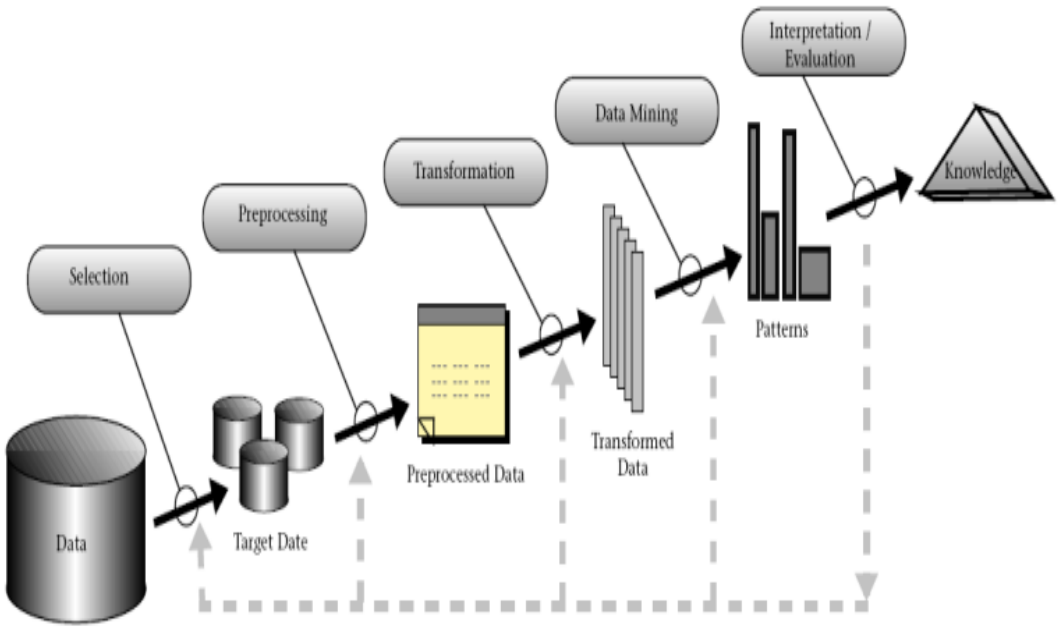

FIG. 1.1 – Des différentes étapes du processus d'ECD

utilisables par l'utilisateur final. Les schémas extraits sont à la fois très nombreux et difficilement compréhensibles pour les non experts du domaine de l'ECD. De ce fait, dans le processus ECD, une phase de post-traitement est proposée pour interpréter des connaissances extraites et les rendre utilisables et compréhensibles. La figure 1.1 décrit le processus d'ECD.

1.2.1 Fouille de données

La fouille de données est considérée comme le coeur du processus ECD. Il s'agit des méthodes et des techniques appliquées aux données pré-traitées pour extraire des schémas et des connaissances. Selon l'objectif de l'utilisateur, il existe différentes approches dont les résultats varient. En général les méthodes de fouille de données sont catégorisées en deux groupes : (1) Supervisées (2) non supervisées. Les approches supervisées nécessitent une phase d'apprentissage et une expertise sur les données. Les approches non supervisées pour lesquelles on ne dispose d'aucune autre information préalable que la description des exemples (enregistrements) [Fio07]. Les méthodes principales de fouille de données sont : la classification, le clutering, l'extraction des règles d'association, de motifs séquentiels fréquents, etc. Nous allons les détailler dans la suite de cette section.

Règles d'association

Les règles d'association ont été crées pour répondre à la problématique du panier de la ménagère. Étant donnée une base de transactions (les paniers), chacune composée d'items (les produit achetés), la découverte de règles d'association consiste à chercher des ensembles d'items fréquemment liés dans **une même transaction**. La recherche de règles d'associations s'avère donc très utile pour la découverte de relations. Le problème des règles d'association est défini dans [AIS93] :

Définition 1.1. Soit ***I***$= \{i_1, i_2, ..., i_n\}$ où i_k = **item**, Soit ***D*** une **base transactionnelle** telle que chaque transaction ***T*** (enregistrements) dans la base est un sous-ensemble de ***I***. Dans une base de données transactionnelle, une transaction se caractérise avec un identifiant et un ensemble non vide d'items.

Définition 1.2. Un itemset est un ensemble non vide et **non ordonné** d'items noté $(i_1, i_2, ..., i_k)$ où i_j est un **item** de ***I***.

Définition 1.3. Une transaction ***T*** de ***D*** supporte ***l'itemSet X*** si elle contient tous les items de ***X***.

Définition 1.4. Une **règle d'association** est une implication de la forme ***X*** $\rightarrow$ ***Y*** où ***X*** et ***Y*** sont un **itemSet** inclus dans ***I*** tel que ***X***$\wedge$***Y***$=\phi$.

Définition 1.5. Le **support** d'une règle d'association correspond au pourcentage de toutes les transactions de ***D*** supportant l'itemSet ***X*** $\wedge$ ***Y***.

Définition 1.6. La **confiance** de la règle d'association est la probabilité qu'une transaction supportant l'itemSet ***X*** supporte aussi l'item(set) ***Y***.

$$conf(R) = p(Y \subseteq T \mid X \subseteq T) \Longrightarrow support(X \cup Y)/support(X)$$

Exemple 1.1. Selon la base d'achats décrite dans le tableau 1.1, on trouve la règle " (Soda, Chips) $\longrightarrow$ (Chocolat) " avec le support égal à 44% et une confiance de 66% ". L'itemset (Soda, Chips) est présent dans 4 transactions sur 9 en même temps que l'item "Chocolat" (Supp=44%), et l'item "Chocolat" apparaît dans 4 transactions parmi les 6 transactions contenant l'itemset (Soda, Chips) (conf=66%).

Motifs séquentiels fréquents

Les motifs séquentiels, présentés dans [AS95], peuvent être considérés comme une extension des règles d'association dans la dimension du temps. En effet, ils mettent en évidence des associations inter-transactions, contrairement aux règles d'association qui extraient des combinaisons intra-transactions. Dans une base d'achats de supermarché, le but est de chercher des motifs séquentiels fréquents identifiant les comportements fréquents dans les achats des clients. Les motifs séquentiels mettent donc en évidence des corrélations entre les transactions. Par exemple, sur la base d'achats (tableau 1.1), on peut identifier des motifs séquentiels comme $\langle(Chocolat, Soda)(gateaux, chips)(minceur)\rangle$. Ce motif séquentiel se traduit ainsi : "Fréquemment les clients achètent d'abord du chocolat et du soda, puis dans une prochain achat, ils achètent des gâteaux et des chips et ensuite ils reviennent plus tard pour acheter des produits minceurs ".

Définition 1.7. Un **motif séquentiel** est une liste **ordonnée** non vide d'**itemsets** notée $\langle S_1 S_2 ... S_p\rangle$ où $S_{j\{j=1,...,P\}}$ est un **itemset**.

Exemple 1.2. Si un client achète les produits *a, b, c, d* et *e* selon le motif séquentiel $S = \langle(a)(b, c)(d)(e)\rangle$, cela veut dire qu'il a acheté le produit *a*, puis les produits *b* et *c* ensemble, ensuite le produit *d* et plus tard le produit *e*.

Définition 1.8. Un motif séquentiel $S' = \langle S'_1 S'_2 ... S'_m\rangle$ est une **sous-séquence de motif séquentiel** $\langle S_1 S_2 ... S_p\rangle$ s'il existe des entiers $a_1 < a_2 < ... < a_j ... < a_m$ tels que $S'_1 \subseteq S_{a_1}$, $S'_2 \subseteq S_{a_2}$, ..., $S'_m \subseteq S_{a_m}$. On dit que S' est **inclus** dans S.

Exemple 1.3. La séquence $\langle(b)(e)\rangle$ est une sous-séquence de S car $(b) \subseteq (b\,c) et (e) \subseteq (e)$, mais $\langle(b)(c)\rangle$ n'est pas sous-séquence de S car $\langle(b)(c)\rangle$ n'est une sous-séquence de $\langle(b\,c)\rangle$.

Clients	Date	Savon	Soda	Chips	Pizza	Farine	Minceur	Gâteau	Chocolat
C_1	01/5/2008		X	X					X
C_2	01/5/2008				X		X		X
C_3	01/5/2008		X	X		X			X
C_1	07/5/2008		X	X	X	X		X	
C_2	07/5/2008	X	X	X	X			X	X
C_3	07/5/2008	X		X		X		X	
C_1	14/5/2008		X				X		X
C_2	14/5/2008	X	X	X			X		
C_3	14/5/2008		X	X			X		X

TAB. 1.1 – Base d'achats des clients

Définition 1.9. Une **séquence de données** est constituée par les transactions regroupées par un client et ordonnées chronologiquement. Un ensemble de toutes les séquences de données est une base transactionnelle.

Exemple 1.4. Sur la base d'achats du tableau 1.1, la séquence de données du client 1 est notée sous la forme :
$\langle(Soda, Chips, Chocolat)(Chips, Pizza, Farine, Gâteau)(Soda, Minceur, Chocolat)\rangle$

Définition 1.10. Un client **supporte** une séquence S si S est incluse dans la séquence de données de ce client. Le support d'une séquence S est calculé comme étant le pourcentage de clients qui supportent S. Soit **minSupp** le support minimum fixé par l'utilisateur. Une séquence qui vérifie le support minimum (*i.e.* dont le support est supérieur à minSupp) est une séquence **fréquente**.

Exemple 1.5. Selon la base d'achats 1.1, le motif $\langle(Chocolat, Soda)(gâteau, chips)(minceur)\rangle$ est inclus dans la séquence de données de client 1 et dans la séquence de données de client 3, donc si minSupp = 50%, le motif est fréquent.

Classification

La **classification** est une méthode supervisée qui consiste à définir une fonction qui attribue une ou plusieurs classes à chaque donnée.

Définition 1.11. Soit un ensemble X de n données labélisées. Chaque donnée x_i est caractérisée par P attributs et par sa classe $c_i \in C$ où C est un ensemble fini. Suite à une phase d'apprentissage sur l'ensemble des exemples $X = (x_i, c_i)_{i\in\{1,\dots,N\}}$, on cherche à prédire la classe de toute nouvelle donnée.
On utilise le label de données comme synonyme de "classe". Un exemple est une donnée pour laquelle on connaît à priori sa classe.

Jour	Ciel	Température	Humidité	Vent	Jouer
1	soleil	chaud	élevée	faible	non
2	soleil	chaud	élevée	fort	non
3	pluie	doux	élevée	faible	oui
4	pluie	doux	élevée	faible	oui
5	soleil	chaud	élevée	fort	?

TAB. 1.2 – Base d'exemples "jouer au tennis" pour la classification

On parle de classification binaire, si l'on classifie les données en deux classes (| C |=2) et n-aire si l'on classifie les données en n classes.

Exemple 1.6. On considère le tableau 1.2 contenant les données de l'apprentissage pour la classification. Chaque instance est labélisée par "Oui" et "Non". La classification consiste à construire un modèle permettant prédire ça marche pour le tennis ? si "ciel=soleil, température=chaud, humidité=élevée et vent=fort". Cette déduction se fait par rapport à l'apprentissage sur le jeu de données.

Clustering

Le **clustering** (**Segmentation**) est une méthode non-supervisée de regroupement des éléments. Contrairement à la classification, il n'y a pas de connaissances a priori sur les classes pré-définies des éléments. La séparation des objets dans les différents groupes (clusters) s'effectue en se basant sur le calcul de similarité entre les éléments. L'objectif des techniques du clustering est de grouper des éléments proches dans un même groupe de manière à ce que deux données d'un même groupe soient le plus similaires possible et que deux éléments de deux groupes différents soient le plus dissemblables possible [Har75]. Cette manière de définir

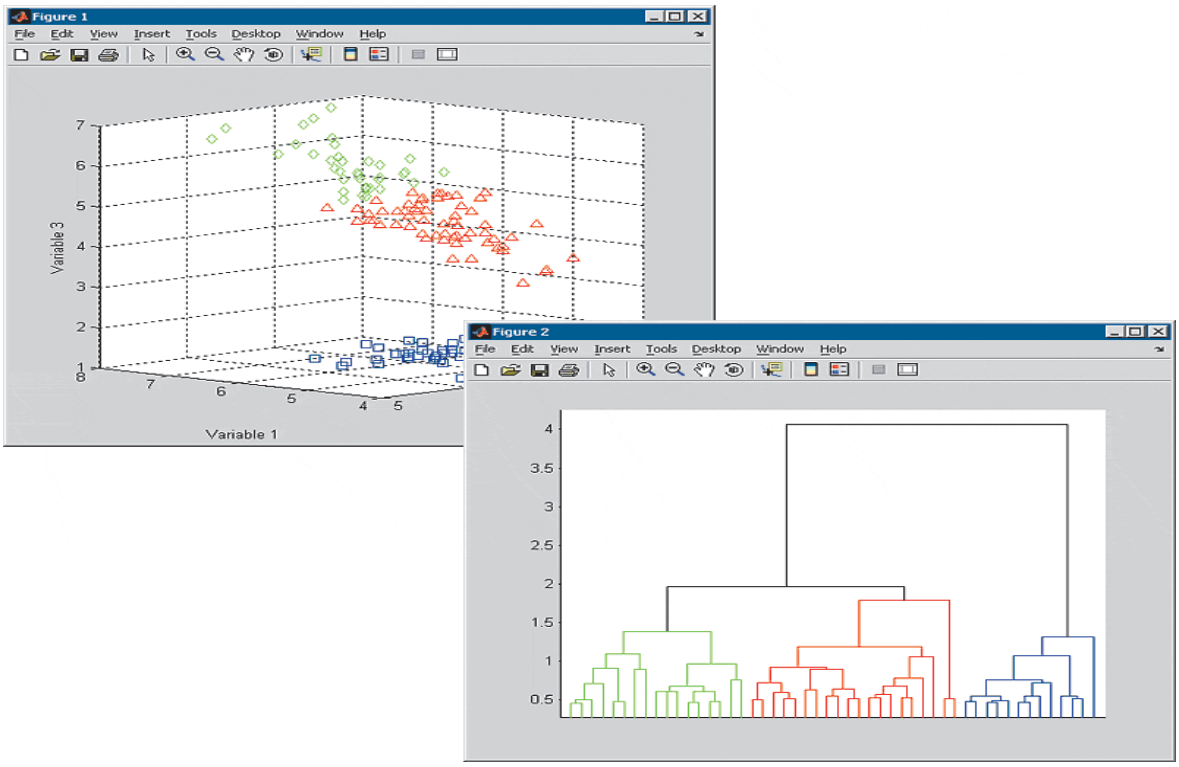

FIG. 1.2 – Clustering non-hiérarchique (en haut) et clustering hiérarchique (en bas)

intuitivement l'objectif de la segmentation cache la difficulté de formaliser la notion de ressem-

blance entre deux éléments. Une partie importante de la définition une méthode de clustering consiste en effet à définir et formaliser une mesure de similarité adaptée aux caractéristiques des données.

Définition 1.12. Soit un ensemble X de N données décrites chacune par leurs P attributs. Le clustering consiste à créer une partition ou une décomposition de cet ensemble en groupes (clusters) telle que :

- les données appartenant au même groupe se ressemblent,
- les données appartenant à deux groupes différents soient peu ressemblantes.

Il existe deux types du clustering : (1) le clustering non-hiérarchique (2) le clustering hiérarchique. Dans le premier cas, on décompose l'ensemble d'individus en K groupes. Dans la 2ème, on décompose l'ensemble d'individus en une arborescence du groupes. Le figure 1.2 montre un exemple du clustering non-hiérarchique. Les éléments sont regroupés dans les clusters selon leurs similarités.

État de l'art

Le monde a deux histoires : l'histoire de ses actes, celle que l'on grave dans le bronze, et l'histoire de ses pensées....

Georges Duhamel

Chapitre 2

Détection d'intrusions

Les **systèmes de détection d'intrusions** (IDS, Intrusion Detection Systems en anglais) sont importants pour la sécurité de réseau. Ils permettent d'identifier les activités internes ou externes qui compromettent la sécurité du réseau et des informations. Les intrusions internes sont faites par un utilisateur interne ayant certains privilèges. Cet utilisateur essaie de faire des activités pour lesquelles il n'a pas de droits. Les intrusions externes sont des activités faites par des utilisateurs inconnus sur le réseau. La détection d'intrusions est un domaine actif. En général, on peut classer les approches en deux groupes : (1) celles basées sur des techniques statistiques (2) celles basées sur des techniques de fouille de données et l'apprentissage. Nous allons dans cette section présenter certains travaux représentatifs de ces deux groupes.

2.1 IDS et méthodes statistiques

Les techniques statistiques connues sous le nom d'apprentissage "top-down", sont employées lorsqu'on possède une connaissance a priori des relations que l'on cherche et que l'on souhaite quantifier [Bru04]. L'Estimateur Bayes, le modèle déviation standard et le Modèle Makov sont des méthodes statistiques utilisées pour la détection d'intrusions et discutées dans [Den87]. Staniford et al [SHM02] utilise réseau Bayesian pour calculer la probabilité de varier des attributs d'une connexion en respectant les attributs d'autres connexions. Dans le cas des approches statistiques, on a des connaissances a priori sur les relations que l'on cherche. Or plus les systèmes d'informations deviennent complexes et grands, plus les intrusions sont elles aussi complexes. Or, on ne dispose pas toujours de connaissances sur les nouvelles méthodes d'attaques. Les systèmes de détection d'intrusions basées sur les techniques de fouille de données sont plus adaptées. Nous allons développer dans la suite les travaux utilisant des techniques de fouille de données.

2.2 IDS et fouille de données

Les méthodes de fouille de données les plus utilisées pour la détection d'intrusions sont la classification et le clustering. Avant de présenter de ces méthodes nous allons détailler une étape commune à ces méthodes : la sélection d'attributs (feature selection en anglais).

2.2.1 Sélection d'attributs

Une choix pertinent des attributs impacte sur la qualité des résultat obtenue. Les méthodes de fouille de données sont plus efficaces s'il existe des connaissances sur les attributs de domaine, sur la priorité de ces attributs, sur les attributs moins importants et les relations éventuelles entre les attributs [L.97].

	Timestampe	IP-Source	IP-Destination	Port-Source	Port-Destination	Protocol	Duration	Source Bytes	Destination bytes	TCP flages	Duplicated ACK rate	% Data packets
Lee et Stolfo(98) [LS98]	X	X	X	X	X	X	X	X	X	X	X	X
Lee et al(99) [LSM99]	X	X	X	X	X		X	X	X	X		
Sinclaire et al(99) [SPM99]		X	X	X	X	X						
Neri(2000) [Ner00]		X	X	X	X	X	X	X		X		
Lee et al(2000) [LSM00]	X	X	X	X	X		X	X	X	X		
Lee et Stolfo(2000) [LS00]	X	X	X	X	X	X	X	X	X	X		
Dickerson(2000) [DD00]	X	X	X	X	X			X	X	X		
Portnoy et al(2001) [PES01]						X	X	X	X	X		
Chittur(2001) [Chi01]					X	X	X	X	X	X		
Lee et al(2001) [LX01]	X	X	X	X	X		X			X		
Lee et al(2002) [LSM]	X	X	X		X		X	X		X		
Chan et al(2003) [CA03]		X	X	X	X		X	X	X	X		

TAB. 2.1 – Attributs intrinsèques utilisés par différents auteurs

Lee et al [LX01] remarquent que un choix d'attributs dont les valeurs changent significativement dans les cas d'anomalies par rapport aux cas généraux est essentiel pour optimiser la détection d'intrusions. Les sources les plus utilisées comme données à analyser sont les logs

de connexion. Le traitement des logs peut se faire soit en ligne, soit hors-ligne. Sur les logs, certains attributs sont des caractéristiques intrinsèques de la connexion, par exemples les attributs IP-Source, IP-destination, Port-Source et Protocole. Le tableau 2.1 présente les attributs intrinsèques et les auteurs qui les citent. Parmi les attributs intrinsèques on trouve les attributs axis : IP-source, Porte-Source, IP-destination, Porte-Source, protocole introduits par [LSM98]. Ce sont des attributs essentiels de chaque transaction. L'auteur note que les relations entre les attributs non-axis ne sont pas intéressantes. Autrement dit, il cherche uniquement des relations dans lesquelles il existe au moins un attributs axis. L'attribut de référence est un attribut dont la valeur est constante pendant le traitement. Par exemple le choix d'IP-destination comme attribut référence permet de découvrir des relations associées à un hôte particulier. Par exemple Dickerson a obtenu de meilleurs résultats en limitant la recherche aux trois attributs nommés : "sdp" (*i.e.* source-IP, destination-IP et destination-port) [DD00]. Les attributs intrinsèques d'une connexion ne fournissent pas assez d'informations pour découvrir des intrusions efficacement. Pour optimiser le choix des attributs on calcule d'autres attributs à partir des attributs intrinsèques [LS98] [LS00] [LSM99] [DD00] [Lue99].

[LS98] augmentent la précision des résultats en insérant des informations temporelles pour avoir de nouveaux attributs. Pour cela, ils définissent les attributs calculés (calculated attributs) qui représentent la moyenne d'un attribut ou le pourcentage des connexions respectant certains critères pendant les dernières w seconds [LS98] [LS00] [LSM99]. La définition des attributs calculés est formalisée dans [LS00] comme des fonctions portant sur d'autres attributs et utilisant un ensemble prédéfinit d'opérateurs comme "le nombre", "le pourcentage" ou "la moyenne", ainsi qu'un ensemble de contraintes comme "le même hôte", "le même service" ou "le time window" (fenêtre temporelle). Le tableau 2.2 montre les principaux attributs calculés utilisés dans chaque citation. La difficulté concernant les attributs calculés est la détermination de valeurs de contraintes. Par exemple, dans [LS00], le time window pertinent sur le jeu de données IES est 30 seconds lorsque dans [LSM00] il est égal à 140 seconds sur le jeu de données IWSS16. Pour améliorer la pertinence de choix d'attributs et par conséquence la performance des traitements, Lee et Xiang [LX01] utilisent des mesures issues de la théorie de l'information pour examiner l'utilité de chaques attributs. Les mesures utilisées sont l'entropie, l'entropie conditionnelle et le gain d'informations.

La méthode de choix d'attributs prédominante (sans considération de la méthode de fouille de données) est basée sur le principe "heuristique-essaie-erreur". Dans la littérature, un bon choix d'attributs consiste en attributs intrinsèques et attributs calculés. Une fois les attributs choisis, il existe des méthodes pour déterminer la pertinence de chaque choix comme des mesures de la théorie de l'information [LX01].

	Lee et Stolfo(98) [LS98]	Lee et al(99) [LSM99]	Lue(99) [Lue99]	Lee et Stolfo(2000) [LS00]	Lee et al(2000) [LSM00]	Dickerson(2000) [DD00]	Chittur(2001) [Chi01]	Portnoy et al(2001) [PES01]	Lee et al(2002) [LS98]	Didaci et al(2002) [DGR]
% of same service to same host	X	X		X	X		X	X	X	X
% on same host to same service				X	X		X	X		X
Bytes transferred current host										X
Bytes transfered current service	X									X
Bytes transferred all services	X				X		X		X	X
Average duration current host										X
Average duration current service	X				X		X	X	X	X
Average duration all service	X				X		X		X	X
nb ICMP packets										X
nb Packets to all services						X				X
nb Different services accessed			X							X
nb To same host		X	X	X	X	X	X	X	X	X
nb To same service	X	X	X	X	X	X	X	X	X	X
nb FIN flags			X		X	X	X	X	X	X
nb RST flags			X		X	X	X	X	X	X
nb SYN flags			X		X	X	X	X	X	X
nb Establishement errors	X	X		X	X	X	X	X	X	X
nb Total connections	X	X		X	X	X	X	X	X	X

TAB. 2.2 – Attributs calculés utilisés par différents auteurs

2.2.2 IDS et Classification

Les IDS basés sur les techniques de classification ont pour but de classifier les trafics d'un réseau en deux classes : "général" et "intrusion". La classification nécessite un apprentissage. La précision de cet apprentissage assure la diminution du taux de faux positifs (*i.e.* les cas normaux classés comme intrusions) et du taux de faux négatifs (*i.e.* les intrusions classées comme normales).
Le système "**RIPPER**" utilise la méthode de "génération des règles inductives"[1] de la classification [LSM99], [LS98], [LS00], [LSM00].
Ce système est efficace pour classifier les cas dans la catégorie normal et dans différents cas d'intrusions (la classification n-aires). Deux caractéristiques de **RIPPER** sont (1) les règles générées par ce système sont faciles à comprendre (2) la possibilité de générer plusieurs ensembles de règles. **REGAL** est un IDS utilisant des techniques de classification basées sur l'algorithme génétique[2] [Ner00]. Cette approche ressemble aux approches basées sur les règles inductives mais l'auteur ne clarifie pas l'efficacité de l'approche.
L'arbre de décision est utilisé comme technique de classification dans [Chi01]. Chaque noeud de l'arbre représente un attribut dans le jeu de données. Les attributs sont pondérés et la décision finale sur le type de connexion dépend des poids des attributs. En parcourant l'arbre de la racine vers les feuilles, on trouve les règles de décisions (constituées des (attributs, valeurs) présents sur les noeuds correspondants) selon lesquelles on classifie les nouvelles instances.
Dans [SPM99], les auteurs convertissent les attributs de connexions de réseau sous la forme d'une séquence de gènes. Chaque connexion est comparée avec l'ensemble des chromosomes. S'il n'existe pas de correspondance, la connexion est étiquetée comme anomalie.
Le problème fréquent dans les approches précédentes est l'utilisation de valeurs précises ou d'intervalles avec des frontières apposées. Ces systèmes ne sont pas assez flexibles et sensibles aux changements des valeurs des attributs. Si l'on a des intervalles avec des frontières apposées, les valeurs hors intervalles sont traitées de la même manière sans prendre en compte la distance de chaque valeur avec les frontières. Par exemple, les valeurs 1 et 9 hors intervalle $[10...20]$ sont traitées de la même manière. Pourtant, le 9 (contrairement au 1) est très proche de la frontière. Une solution pertinente pour résoudre ces problèmes est l'utilisation de la logique floue pour le traitement des données numériques qui permet d'obtenir des résultats plus abstraits et plus flexibles. La logique floue s'appuie sur la théorie mathématique des ensembles flous. Cette théorie, introduite par [Zad65], est une extension de la théorie des ensembles classiques pour la prise en compte d'ensembles définis de façon imprécise. C'est une théorie formelle et mathématique utilisant le concept de degré d'appartenance pour modéliser la définition d'un sous-ensemble

[1] Une règle d'inductive est logiquement l'équivalent d'un ensemble infini de formules, qui justifient l'utilisation de la preuve par induction.
[2] Meta-heuristiques permettant obtenir une solution approchée en un temps correct

dans un univers donné. [Zad65] a aussi montré que cette théorie se réduit effectivement à la théorie des sous-ensembles classiques dans le cas où les fonctions d'appartenance considérées prennent des valeurs binaires (0,1).
[DD00] combine la logique floue et la classification pour traiter des portions de données (temporellement liées) pour les classer en deux catégories général et intrusion. Cette approche est efficace pour détecter les intrusions de type SCAN : le réseau est scanné pour connaître l'architecture et découvrir des points vulnérables. Un utilisateur se connecte à plusieurs hôtes successivement pour une courte durée. Par conséquent, il y a des sessions de connexion qui sont reliées. Vu la classification se fait sur des portions reliées temporellement, cette approche a de bons résultats pour détecter les SCANs. L'inconvénient est la tâche difficile de génération des règles et de définition d'une bonne contrainte (time-window) pour déterminer les portions.
[Lue99] a développé le travail de Lee et al [LS98], [LSM99] en ajoutant la notion de logique floue. Il a obtenu des résultat plus flexibles. Selon lui, la détection d'intrusions est une application naturelle de la logique floue car déterminer de manière absolue qu'une connexion donnée est une intrusion ou pas n'est pas possible. Avec la logique floue, on peut donner la probabilité d'une intrusion. [Lue99] constate une bonne réduction de taux de fausses alertes. [Lue99] obtient des résultats du types : " X est une anomalie et la probabilité que X soit normal est basse".

2.2.3 IDS et Clustering

Le clustering est une technique de fouille de données permettant de regrouper les éléments selon leur ressemblance (cf. la section 1.2). Contrairement à la classification, il n'y a pas besoin de connaissances a priori sur les classes dans lesquelles on regroupe les éléments. On n'a pas besoin des données labelisées (par les classes prédéfinis) pour l'apprentissage.

Pour détecter des intrusions, [PES01], [CA03] cherchent des outliers[3]. Ils appliquent l'algorithme des K-Plus Proches voisins (K-Nearest Neighbors) sur des logs de connexion. Les outliers trouvés représentent des intrusions. [CA03] note que l'on retrouve des attaques dans les clusters (groupes) ayant une densité élevée ou basse statistiquement. C'est pourquoi il a orienté le clustering selon la densité et la distance des clusters en même temps. Dans [BCH+01], les auteurs utilisent l'algorithme k-means pour la quelle il faut préciser a priori le nombre de clusters (k). Les éléments sont donc regroupés dans K clusters. La détermination du nombres de clusters est délicate et la performance de clustering en dépend. Si K est très grand, certains clusters pourraient former potentiellement un seul cluster. Si K est petit, les éléments non ressemblant seront regroupés dans un seul cluster. Dans [SZ02] les auteurs ont donc développé une méthode de clustering dynamique où le nombre de clusters est déterminé pendant le clustering. Si on

[3] Observation numériquement très différentes du reste des données

	Approche	Algorithme	(+)	(-)
Lee et Stolfo(98) [LS98]; Lee et al(99) [LSM99]; Lee et Stolfo(00) [LS00]; Lee et al(00) [LSM00]; Lee et al(01) [LX01]	Classification	règles d'association pour trouver les relations entre les attributs; Règles Inductives	Axis attributs, Attributs de référence; Attributs Calculés; Induction des règles rapide	Taux de fausses Alertes; Grande ensemble d'attributs
Sinclair(99) [SPM99]	Classification	Arbre de décision	Précision de la classification des connexions	Manque de la comparaison
Lue(99) [Lue99]	Classification	Logique floue; règles d'association floue;	Amélioration du RIPPER (Lee et al) en utilisant la logique floue; Bon résultat pour détecter les jeu de données ayant anomalies; règles plus flexibles	Non résultats données sur la détection des anomalies aux-même;
Neri(00) [Ner00]	Classification	règles d'association; Algorithme Génétique	Compression des attributs pour diminuer l'occupation de la mémoire; Précision des règles extraites	Sans remarque sur la clarté ou éfficacité des règles;
Dickerson(00) [DD00]	Classification	Logique floue; Mesures statistiques	Partitionner le jeu de données en partions liées temporellement; Bon résultats pour les intrusions SCAN	Création manuellement; Efficace seulement pour les cas comme SCAN;
Chittur(01) [Chi01]	Classification	Arbre de décision		Le taux de la détection bas;
Toutes ces approches ont besoins de données labélisées par les classes "normal" et "intrusion". Or labélisation n'est pas une tâche facile surtout sans expert				

TAB. 2.3 – Approches utilisant la classification dans le domaine IDS

Approche	Algorithme	(+)	(-)
Sequeira et Zaki(02) [SZ02]	Clustering et Clustering	Dynamique clustering	Détection des masqueraders ; Diminution du taux de fausses alertes ; Profiling par clustering	Host-based IDS
Chan et al(2003) [CA03]	Clustering et Clustering	Détection d'outliers par rapport au clusters, KNN clustering	Clustering par rapport à la densité des clusters	

TAB. 2.4 – Approches utilisant le clustering dans le domaine IDS

ne peut pas mettre un élément dans un cluster (à cause de la contrainte de similarité), on crée un nouveau cluster. La méthode **ADMIT** [SZ02] est un IDS fondé sur le clustering dynamique. ADMIT est un système IDS permettant de distinguer les utilisateurs réels d'un système Unix et les masqueraders (qui essaient d'abuser de système en utilisant le compte d'un utilisateur réel). Un profil est créé pour chaque utilisateur du système selon les séquences de commandes utilisées sur la console. En faisant le clustering dynamique sur les séquences de commandes, on crée des groupes de commandes. Ces groupes sont raffinés à cause du bruit dans les données. Dans la phase d'évaluation, une comparaison de la similarité entre les séquences de commandes et les centres des clusters détermine si la séquence de commandes appartient à l'utilisateur réel ou à un masquerader. ADMIT permet de trouver les intrusions sur un hôte particulier. Il ne permet pas trouver les intrusions au niveau du réseau et du domaine.

2.3 Discussions

Il existe des travaux fondés sur les techniques de fouille de données dans le domaine de la détection d'intrusions. Nous avons montré dans la section 2.2.1 que le choix des attributs impacte sur la qualité des résultats. Ce choix dépend de l'objectif de la détection d'intrusions et de méthodes utilisées. La plupart du temps, une combinaison d'attributs intrinsèques et d'attributs calculés est pertinente. Nous notons que cette tâche, nécessite une exploitation détaillée de logs des réseaux. par exemple, Lee [LS98] note qu'il a réussi à extraire environs 40 attributs à partir de logs. Les tableaux 2.1 et 2.2 montrent les attributs intrinsèques et les attributs calculés avec les citations utilisées de chaque ensemble d'attributs. Les système de détection d'intrusions s'améliorent mais il n'existe pas encore une solution répondant à tous problèmes notamment le taux élevé des fausses alertes. De plus, la moitié des approches identifie seulement un type particulier d'intrusion comme [DD00] pour les intrusions SCAN ou [SZ02] pour les masqueraders sur un hôte particulier. Les tableaux 2.3 et 2.4 représentent les travaux dans le domaine d'IDS en montrant les méthodes utilisées avec leurs avantages et les inconvénients de chaque approche. L'efficacité des systèmes de la détection d'anomalie, au niveau de fausses alertes dépend quasiment sur la manière de modéliser les comportements normaux. C'est la raison pour la quelle, nous proposons le clustring de motifs séquentiels pour la modélisation. D'autre part, étant donné de la principe de la détection d'anomalie (contrairement de la détection d'abus) nous permet de trouver les intrusions non pas encore connues. La différence entre la méthode ADMIT et notre proposition pour la système IDS consiste en 2 parties :

- L'objectif de notre proposition est de trouver toutes les intrusions éventuelles au niveau réseau (domaine).
- Pour créer des profils, nous utilisons des motifs séquentiels fréquents. Il existe très peu

de travaux sur l'utilisation de motifs séquentiels dans le domaine d'IDS et surtout pour la création de profils de comportements.

La caractéristique des motifs séquentiels (corrélations entre les événement séquentiels au fil du temps) nous permet de modéliser les comportements généraux des utilisateurs d'un réseau sur une durée précise. Une déviation par rapport à ces profils normaux est alors une intrusion (principe de la détection d'anomalie). Notre solution est capable de trouver les périodes temporelles dans lesquelles il y a des intrusions et les intrusions elles-mêmes. Cela dépend du format du jeu de données utilisé lors de l'extraction de motifs séquentiel et les solutions utilisées pour définir les notions le "client" et la "transaction".

Le clustering de motifs séquentiels est alors indispensable. Il existe très peu de travaux sur la similarité et le clustering des motifs séquentiels (séquence d'itemset). C'est pourquoi nous introduisons une mesure de similarité adaptée pour les motifs séquentiels qui pourra être utilisée pour d'autres applications que le clustering.

Dans la suite, nous allons détailler les travaux existant sur les mesures de similarité pour les séquences d'items et les séquences d'itemsets (motifs séquentiel).

Chapitre 3

Mesure de similarité pour les motifs séquentiels

Afin de modéliser les comportements des utilisateurs, nous proposons une approche basée sur les motifs séquentiels (définition cf. section 1.2). Les profils de comportement est crées par un clustering des motifs séquentiels. Pour cela, nous avons besoin de comparer leur similarité. Or, il existe peu de travaux sur la comparaison des motifs séquentiels.
Dans la littérature, on trouve plusieurs mesures de similarité pour comparer des séquences d'items. Malheureusement, ces techniques ne sont pas adaptées aux séquences d'itemsets (motifs séquentiels). C'est pourquoi nous définissons une mesure de similarité pertinente pour les motifs séquentiels. Dans ce chapitre, nous détaillons les travaux existant sur les mesures de similarité pour les séquences. Dans la section 3.1, nous présentons les mesures existant pour les séquences d'items et nous expliquons pourquoi ces mesures ne sont pas adaptées aux séquences d'itemsets (motifs séquentiels). Dans la section 3.2, nous détaillons les travaux portant de la comparaison de motifs séquentiels et également des travaux sur le clustering de motifs séquentiels. Nous expliquons les raisons pour lesquelles ces mesures ne sont pas pertinentes. Notre mesure sera détaillée et formalisée dans le chapitre 5.

3.1 Mesures de similarité pour les séquences

La Similarité est un degré (symétrie ou non-symétrie[1]) de la ressemblance entre deux ou plusieurs concepts ou objets. La notion de similarité repose soit sur des répétitions exactes ou des approximation de motifs dans les deux items comparés (c.f. WikiPedia).
Dans certains domaines, comme la biologie, le traitement des logs, la détection d'anomalies,

[1] il existe des mesures de similarité non-symétriques

le traitement des données commerciales, le traitement de la langue naturelle, la télécommunication, etc les données peuvent être considérées sous forme des séquences. Le clustering de séquences, la détection d'outliers, le traitement du langue naturel, la visualisation des données séquentielles, l'extraction de séquences fréquentes sous contrainte de similarité sont alors des applications pour lesquelles la comparaison de la similarité des séquences[2] est utile. Dans certaines applications et méthodes comme la bio-informatique, l'extraction de motifs séquentiels sous contrainte de similarité etc, on cherche les séquences similaire à une séquence référence (cible). Une mesure de similarité non-symétrique peut être utilisé quand on compare les deux éléments selon une direction (e.g. la comparaison des séquences données avec une séquence de référence). La définition de la similarité peut varier selon le type de similarité que l'on cherche. Les mesures de similarité différentes peuvent refléter les différents visages de données. Deux objets peuvent être considérés très similaires par une mesure et très différents par une autre mesure [Moe00]. Afin de comparer deux séquences, il est nécessaire de pourvoir calculer la similarité ou la distance entre les séquences. Plus le degré de similarité (distance) entre deux séquences est élevé, plus les deux séquences se ressemblent (sont loin l'un de l'autre). Plusieurs approches ont été développées pour comparer la similarité entre deux séquences notamment dans le domaine bio-informatique.
Les distances et les mesures de similarité les plus connues entre deux séquences sont :

- **la mesure de cosinus**,
- **la distance euclidienne**,
- **la distance de Hamming**,
- **l'Edit Distance** (**distance de Levenshtein**),
- **MCP** (Match Count Polynomial Bound),
- **LCS** (Longest Common Subsequence).

La distance euclidienne et la mesure de cosinus : On considère une séquence composée de n items comme un vecteur de n-dimensions. Le **cosinus** fait l'angle entre des vecteurs normés par leur longueur :

$$cos(V_1, V_2) = \frac{V_1.V_2}{|V_1||V_2|}$$

Dans un espace à deux dimensions, le **cosinus** se calcule de la façon suivante :

[2]Les mesures de distance, utilisées pour comparer la différence de deux éléments représentent le niveau de différence entre deux éléments contrairement à la similarité qui montre la ressemblance. Cela veut dire que si deux objets sont similaires la distance entre eux est petite et vise versa. Les distances sont normalement connues sous le nom "métriques".

$$\cos(V_1, V_2) = \frac{a_1 b_1 + a_2 b_2}{\sqrt{a_1^2 + a_2^2} + \sqrt{b_1^2 + b_2^2}}$$

On peut calculer également la distance entre deux vecteurs normés au lieu de la calculer entre deux vecteurs. Pour ça, on utilise la **distance euclidienne**, mais en normalisant les vecteurs par leur longueur. Les vecteurs normés ont tous une longueur égale à un. Pour deux vecteurs $V_1(a_1, ..., a_n)$ et $v_2(b_1, ..., B_n)$, la distance euclidienne se calcul ainsi :

$$\sqrt{\sum_{i=0}^{n} \mid a_i - b_i \mid^2}$$

La mesure cosinus et la distance euclidienne permettent calculer la similarité (distance) entre deux vecteurs à n dimensions. Ces métriques sont fréquemment utilisées en fouille de textes.

La distance de Hamming : Elle permet de quantifier la différence entre deux séquences de symboles. Cette distance est utilisée dans plusieurs disciplines comme la théorie de l'information, la théorie des codes et la cryptographie.
Soit A un alphabet et F l'ensemble des suites de longueur n à valeur dans A. La distance de Hamming entre deux éléments a et b de F est le cardinal de l'ensemble des images de a qui diffèrent de celles de b (c.f. WikiPedia).

$$\forall\, a, b \in F \quad a = (a_i)_{i \in [1...n]} \text{ et } b = (b_i)_{i \in [1...n]} \quad dis_{Hamming}(a, b) = \mid i \mid_{a_i \neq b_i}$$

Exemple 3.1. La distance de Hamming entre les deux séquences $\{AB\mathbf{A}BBC\mathbf{E}A\mathbf{D}\}$ et $\{AB\mathbf{B}BBC\mathbf{D}A\mathbf{E}\}$ est égale à 3 car ils existent trois éléments différents. Pour comparer des séquences de longueurs variables, ou des chaînes de caractères pouvant subir non seulement des substitutions, mais aussi des insertions ou des effacements, des métriques plus sophistiquées comme la distance de Levenshtein sont plus adaptées.

L'Edit distance : La **distance de Levenshtein** ou **Edit distance** est égale au nombre minimal de caractères qu'il faut modifier pour passer d'une séquence à une autre. La transformation d'une séquence en une autre séquence se fait par une séquence d'opérateurs de modification (Edit operators) appliqués sur la séquence. L'ensemble des opérateurs autorisés, dépend du domaine, de l'objectif et du type de séquences. Les opérateurs fréquemment utilisés sont l'insertion, la suppression et la substitution. Tous ces opérateurs prennent en compte la position des éléments dans la séquence.

Exemple 3.2. Soit séquences $S_1 = \{A, B, A, C, B, D\}$ et $S_2 = \{A, B, C, C, A, D\}$ à comparer. La séquence des opérations (Edit operations sequence) est :

$$O = \{SuppA_3, Ins(C_4), Supp(B_5), Ins(A_5)\}$$

On peut également associer aux opérations un poids. La distance sera le coût de la transformation d'une séquence avec le nombre minimum d'opérations.

MCP **(Match Count Polynomial Bound) :** Cette mesure compte le nombre de positions (slots) identiques dans deux séquences [Lan00].

Exemple 3.3. Pour les deux séquences $S_1 = \{A, C, \mathbf{F}, G\}$ et $S_2 = \{C, G, \mathbf{F}, A\}$, MCP est égale à "1" car il existe seulement une seule position identique (**F** situé à la 3ième position dans les S_1 et S_2).

LCS **(Longest Common Subsequence) :** [SZ02] donne la longueur de la sous-séquence la plus longue commune entre deux séquences.

Exemple 3.4. Sur l'exemple précédant, $LCS(S_1, S_2) = 2$ car la sous-séquence la plus longue $\{C, G\}$ a pour longueur = 2.
Les valeurs des mesures MCP ou LCM sont incluses dans l'intervalle $[0, l]$ où l=longeur des séquences, contrairement aux mesures comme Edit distance normalisée où la mesure de Cosinus avec des valeurs entre $[0, 1]$[3].

Les mesures de similarité précédentes sont adaptées pour calculer la similarité (Distance) entre deux séquences d'items. Une séquence d'items est définie comme une liste ordonnée d'items. Contrairement à un ensemble, l'ordre importe. Un même élément peut apparaître plusieurs fois à différentes positions dans la séquence. En effet, on ne peut pas traiter un motif séquentiel (séquence d'itemset) comme une séquence d'items.
Un motif séquentiel est une liste ordonnée d'itemsets mais non celle des items (l'ordre n'existe pas au sein des items dans un itemset cf. section 1.2). Par conséquent, les éléments d'un motif séquentiel (itemsets) ne sont pas atomiques mais ce sont des ensembles d'items. Cela veut dire que la comparaison des séquences d'itemsets doit se faire au niveau de la séquence et également au niveau des itemsets. Nous allons reprendre ces notions par des exemples suivants :

Problème : Soit deux motifs séquentiels $M_1 = \{(ab)(c)(xy)(d)\}$ et $M_2 = \{(abc)(ad)(abg)\}$. Chaque position dans un motif séquentiel est occupée par un itemset et non un item. Par exemple la 2ième place dans M_1 est (c) et non pas b. Ainsi, les itemsets de M_1 sont (ab), (c), (xy) et (d) et pour M_2 (abc), (ad) et (ab). Appliquons maintenant, les mesures précédentes à ces deux motifs séquentiels.

Exemple 3.5. $Cosinus(M_1, M_2) = 0$, il n'existe pas de dimension commune entre M_1 et M_2. Ce résultat est aussi valide pour la *distance euclidienne*.

Exemple 3.6. $Dist_{Hamming} = 3$, il a trois positions différentes sur les M_1 et M_2

[3] D'après la définition, la valeur 1 représente la ressemblance complète et la valeur 0 aucune ressemblance

Exemple 3.7. $MCP(M_1, M_2) = 0$, il n'y pas de position identique entre M_1 et M_2.
En appliquant les mesures MCP et $Dist_{Hamming}$, on perd la sémantique des motifs séquentiels. Par exemple, la mesure MCP nous dit qu'il n'y a aucune position similaire dans M_1 et M_2. Or, les itemsets se situant dans la 1ère position (i.e. (ab) de M_1 et (abc) de M_2) ne sont pas tout à fait différents ni tout à fait similaires. C'est pourquoi les mesures adaptées aux séquences d'items ne sont pas pertinentes pour les séquences d'itemsets (motifs séquentiels).
En effet, les éléments d'un motif séquentiel (*i.e.* itemset) n'est pas réellement et sémantiquement un élément atomique (comme un item). Pour comparer deux motifs séquentiels de manière pertinente, il faut prendre en compte ces caractéristiques particulières. On ne peut donc pas appliquer les mesures proposées pour les séquences sans les modifier et les adapter aux motifs séquentiels. **LCS** peut être une solution envisageable pour comparer la similarité des motifs séquentiels sans être une solution parfaite et pertinente. Dans littérature, la distance **Edit Distance** est utilisée pour comparer la similarité de deux motifs séquentiels. Nous allons détailler cette mesure et la mesure LCS dans la section suivante (section 3.2). Nous expliquons les raisons pour lesquelles, l'utilisation de ces mesures pour les motifs séquentiels n'est pas une solution pertinente notamment de point de vue la sémantique des motifs séquentiels.

3.2 Mesure de similarité pour les motifs séquentiels

L'extraction de motifs séquentiels fréquents est une méthode puissante dans le domaine de la fouille de données lorsque l'on cherche les événements séquentiels fréquents dans des intervals temporels définis. Malgré l'importance des motifs séquentiels et leurs applications dans plusieurs domaines, il existe peu de travaux sur le clustering de motifs séquentiels et donc sur la comparaison de motifs séquentiels du point de vue de la similarité.
Une mesure de similarité adaptée aux motifs séquentiels, pourrait être appliquée à d'autres applications comme la détection d'outliers dans une base séquentielle, la compression de motifs séquentiels et l'extraction de motifs séquentiels sous contraintes de similarité.
Dans notre contexte, nous allons utiliser la mesure de similarité pour le clustering afin de détecter des anomalies. Nous nous intéressons donc aux approches présentant une mesure de similarité pour les motifs séquentiels ainsi qu'aux méthodes de clustering de motifs séquentiels.

LCS et Motifs séquentiels La mesure LCS (Longest Common Subsequence) sert à la comparaison des séquences [SZ02][4]. Cette mesure donne le longueur de la sous-séquence la plus longue commune à deux séquences. LCS peut être envisageable pour comparer la similarité de

[4] [SZ02] ne propose pas l'utilisation de LCS pour le cas motifs séquentiels.

motifs séquentiels (séquence d'itemset) sans être optimale.
Nous expliquons les raisons pour lesquelles la mesure LCS n'est pas pertinente pour utiliser comme une mesure de similarité. Dans la définition de notre mesure de similarité, nous avons résolu ces inconvénients. Nous notons quatre problèmes avec la mesure LCS.

- La valeur de LCS n'est pas utilisable sans normalisation. En effet la sortie de LCS est incluse dans $[0, l]$ l=le longueur de la séquence. Selon les définitions la valeur d'une mesure de similarité ou de la distance doit être prise entre 0 et 1. Pour normaliser LCS, quelques solutions sont envisageables comme la division par la moyenne du nombre des items.

- LCS n'est pas influencé par la distance entre les itemsets ressemblant. Les itemsets ressemblant dans M_1 et M_2 sont soit consécutifs (dans l'ordre de la séquence), soit non consécutifs (il existe d'autre(s) itemset(s) entre eux).

  Exemple 3.8. Soit les deux motifs séquentiels $M_1 = \{(ab)(c)(xy)(d)\}$ $M_2 = \{(bce)(ad)(abf)\}$. Alors $LCS = 2$, la sous-séquence commune la plus longue est $\{(c)(d)\}$.
  La sous-séquence $\{(c)(d)\}$ n'est pas une sous-séquence avec des itemsets consécutifs dans $M_1 = \{(ab)(c)(xy)(d)\}$. Il existe un itemset (xy) entre les itemsets (c) et (d). Par contre, on trouve la sous-séquence $\{(c)(d)\}$ consécutivement dans $M_2 = \{(b\mathbf{c}e)(a\mathbf{d})(abf)\}$. Évidemment, LCS ne prend pas en compte ce fait mais sémantiquement l'apparition de la sous-séquence $\{(c)(d)\}$ dans M_1 n'est pas similaire à son apparition dans M_2. C'est pourquoi nous considérons un score de l'ordre dans notre mesure qui donne une valeur selon l'ordre des itemsets ressemblant et la distance d'apparition (dans l'ordre de séquence) entre les itemsets (cf. les formules 5.6, 5.7 dans la section 5).

- LCS cherche la longueur de la sous-séquence commune la plus longue entre deux séquences, sans considérer la longueur de la partie qui n'est pas commune.

  Exemple 3.9. Soit $M_1 = \{(ab)(c)(xy)(d)\}$, $M_2 = \{(bce)(ad)(abf)\}$ et $M_3 = \{(ab)(rt)(yu)(ze)(c)(xy)(d)\}$. $LCS(M_1, M_2) = 2$ et $LCS(M_1, M_3) = 2$ avec la sous-séquence = $\{(c)(d)\}$. Or, M_2 ressemble plus à M_1 que M_3. Cela vient du fait que la valeur de LCS n'est pas normalisée par le nombre d'items.

- Dans la sous-séquence commune, le nombre d'items différents dans les itemsets correspondant n'influence pas la valeur de LCS.

Exemple 3.10. Supposons $M_1 = \{(ab)(c)(xy)(d)\}$, $M_2 = \{(bce)(ad)(abf)\}$ et $N = \{(c)(be)(ad)(abf)\}$. $LCS(M_1, M_2) = 2$ et $LCS(M_1, N) = 2$ avec la sous-séquence = $\{(c)(d)\}$.
$LCS(M_1, M_2)$ et $LCS(M_1, N)$ sont égaux cependant l'itemset (c) de la sous-séquence est inclus dans l'itemset (bce) de M_2 lorsque dans N, il est inclus dans l'itemset (c). Par conséquent, M_1 est plus similaire à N que à M_2. On n'a pas pris en compte le fait que dans l'itemset (bce), ils existent deux autres items différents de "c", ce qui n'est pas le cas d'itemset (c) de N. Ce problème n'est pas résolu avec la normalisation comme le cas précédant. Ici, le nombre d'items dans M_2 et N est égal (*i.e. 8*). La différence vient du nombre d'items non communs aux itemsets correspondant.
Dans notre mesure, pour chaque couple d'itemsets ressemblant, nous donnons un poids de similarité (c.f Poids du Mapping dans la section 5) qui prend en compte le nombre d'items non communs entre deux itemsets ressemblant. Par exemple, dans l'exemple précédant, La similarité de M_1 et $M2$ n'est pas égale à celle de M_1 et N.

Edit distance et motif séquentiel La mesure Edit distance à été utilisée dans [CMB02] dans le cadre de la proposition d'une méthode d'extraction de motifs séquentiels sous contraintes de similarité. La mesure sert à comparer la similarité entre un motif séquentiel candidat et un motif de référence. Les auteurs n'ont pas cherché à adapter la cette mesure (Edit distance) aux caractéristiques particulières des motifs séquentiels. Nous allons expliquer pourquoi l'utilisation d'Edit distance comme mesure de similarité entre deux motifs séquentiels n'est pas pertinente. Les auteurs définissent un motif séquentiel comme une liste ordonnée de symboles appartenant à $\sum$ où $\sum$ est un ensemble fini d'alphabets. Nous montrons pourquoi cette mesure n'est pas pertinente pour les motifs séquentiels en prenant un exemple :

Exemple 3.11. Soit **Symbole** $\equiv$ **Itemset** (d'après la définition dans [CMB02]). Soit deux motifs séquentiels $M_1 = \{(ab)(c)\}$ et $M_2 = \{(a)(c)\}$ représentés ainsi :

$M_1\{(ab)(c)\} \Rightarrow X \rightarrow Y \mid X = ab, Y = c$ (X, Y = symboles $\in \sum$)
$M_2\{(a)(c)\} \Rightarrow Z \rightarrow Y \mid Z = a, Y = c$ (Y, Z = symboles $\in \sum$)

Comme les opérateurs d'Edit distance sont appliqués sur les symboles (*i.e. itemsets*), la comparaison sera le coût de substitution de $(X, Z, 1)$. Ici, les itemsets (ab) et (a) sont vus comme des événements et les items comme les attributs d'événement. Par conséquent les itemsets (ab) et (a) sont traités comme deux symboles (événement) complètement différents. Toutefois (ab) et (a) ne sont pas deux

symboles (événements) complètement différents mais au contraire, ce sont deux comportements ressemblant. La différence entre M_1 et M_2 n'est sémantiquement pas la substitution des itemsets (ab) et (a) mais simplement l'insertion d'un item b au 1er itemset de la deuxième séquence.

En effet, dans ces travaux, le motif séquentiel est considéré comme une séquence d'événements[5]. Bien qu'un motif séquentiel puisse être vu comme une séquence d'événements, cette interprétation de l'itemset comme un événement n'est toujours pas pertinente. Revenons à la définition des séquences d'événements.

Définition 3.1. Soit $R = \{A_1, A_2, ..., A_m\}$ un ensemble d'**attributs d'événements**. Un événement est un **(m+1)-tuple** $(a_1, ..., a_m, t)$ où $a_i \in Domaine(A_i)$ et t est un nombre réel représentant le **temps d'occurrence** (occurrence time) d'événement [Moe00].

Définition 3.2. Une séquence d'événements est une liste ordonnée d'événements selon l'augmentation du temps d'occurrence [Moe00].

Exemple 3.12. Soit $R = \{cap_1, cap_2, ..., cap_m\}$ un ensemble de capteurs dans un système d'alarme automatique. Une alarme (événement) se produit par rapport aux valeurs des capteurs dans un temps précis. Par exemple, $Alarme_{\mathbf{A}}$ est caractérisée par les valeurs de capteurs : $Alarm_{\mathbf{A}} = cap_1 = 0, cap_2 = 1, cap_3 = 0$. Si par exemple, la valeur du capteur cap_3 devient 1, on aura une autre Alarme $Alarme_{\mathbf{B}}$ (événement) différente de l'$Alarme_{\mathbf{A}}$.

Exemple 3.13. $E = \{(Alarme_{\mathbf{A}}, 9), (Alarme_{\mathbf{B}}, 12), (Alarm_{\mathbf{B}}, 15), (Alarm_{\mathbf{C}}, 20), (Alarm_{\mathbf{A}}, 23)\}$ est une séquence d'événements qui peut être représentée avec les types d'événements : $E = \{A, B, B, C, A\}$.
Nous voyons que les événements $A = (cap_1 = 0, cap_2 = 1, cap_3 = \mathbf{0})$ et $B = (cap_1 = 0, cap_2 = 1, cap_3 = \mathbf{1})$ sont deux événements complètement différents par la valeur du capteur cap_3. Malgré une petite différence dans les attributs (valeur du capteur cap_3), mais selon les données et contexte, A et B sont deux événements complètement différents.

Exemple 3.14. À l'inverse, Imaginons un motif séquentiel :
$M = \{(chips, soda, pains)(pizza)(chips, soda, chocolat)(farine)\}$ extrait de la base d'achats des clients d'un supermarché. Les deux itemsets $(chips, soda, pains)$ et $(chips, soda, chocolat)$ sont différent au niveau d'un seule item.
Si on considère que chaque itemset, il s'agit d'un événement comme dans les séquence d'événements, les deux itemsets $(chips, soda, pains)$ et $(chips, soda, chocolat)$ sont traités comme

[5] Edit distance est fréquemment utilisée pour les séquences d'événements [Moe00]

deux comportements (événements) tout à fait différents. Or, selon le contexte, on sait qu'ils s'agissent des comportements très proches d'un client. On ne peut donc pas les traiter comme deux événements tout à fait différents.
D'après ces exemples, on ne peut pas traiter tous les motifs séquentiels comme des séquences d'événements où chaque itemset est vu comme un événement. Ce type de traitements des motifs séquentiels dépend du contexte et des attributs et du sens des itemsets dans le contexte. Pour une comparaison pertinente et sémantiquement correcte, il faut comparer deux motifs séquentiels au niveau des itemsets dans des séquences et également au niveau des items dans des itemsets. Avec des mesures comme Edit distance qui traitent un itemset comme un symbole (événement) atomique (sans les traiter au niveau des items contenus) ne sont sémantiquement pas pertinente cependant qu'ils sont applicables syntaxiquement.

Dans [PLT07], l'auteur propose une comparaison de la similarité entre deux motifs séquentiels multidimensionnels pour la détection d'outliers dans le domaine des motifs séquentiels multidimensionnels. La distance entre deux séquences multidimensionnelles est définit ainsi : Soient $S_1 = \{b_1, b_2, \ldots, b_k\}$ et $S_2 = \{b'_1, b'_2, \ldots, b'_k\}$ deux séquences multidimensionnelles, $dist$ une mesure de distance et Op un opérateur d'agrégation. La distance entre S_1 et S_2 se définit de la façon suivante :
$d(s1, s2) = Op(dist(bj, b'j)\ pour j = 1 \ldots k$. La comparaison se fait entre des blocs[6] correspondants. Cela veut dire que l'on compare le bloc i de S_1 avec le bloc i de S_2. Ce type de comparaison que l'on retrouve chez d'autres auteurs, n'est pas utile lorsqu'il y a par exemple un décalage dans l'une des séquences.

Exemple 3.15. Soit les deux séquences $S_1 = \{(ab)(d)(c)\}$ et $S2 = \{(z)(ab)(d)\}$. Si on compare seulement les itemsets (blocs) correspondants (*(ab) avec (z), (d) avec (ab), (c) avec (d)*) on ne trouve aucune similarité entre les deux séquences. Par contre en comparant chaque itemset de S_1 avec tous les itemsets de S_2, on trouve la similarité entre les itemsets $(S_1(1), S_2(2))$ et $(S_1(2), S_2(3))$. La comparaison n'est effectuée que sur les itemsets correspondants elle n'est pas pertinente pour trouver la similarité entre deux motifs séquentiels.

3.3 Clustering de motifs séquentiels

Dans [GK01], les auteurs proposent un nouvel algorithme pour le clustering des séquence de données (data-sequence). L'algorithme développé utilise une méthode de clustering basée sur la méthode des K-means. Le principe de la méthode est de trouver un ensemble de caractéristiques (features set), puis de projeter les séquences de donnés dans un espace vectoriel dont les

[6] Ici, Un bloc de données peut être vu comme un itemset

dimensions sont ces caractéristiques. Ensuite, on utilise une méthode de clustering classique comme K-means pour regrouper les séquences de données projetées dans l'espace vectoriel. La phase la plus importante est celle de la sélection des caractéristiques (dimensions de l'espace vectoriel). Pour cela, ils utilisent des motifs séquentiels extraits à partir de séquences de données. Ces motifs séquentiels sont raffinés selon certaines contraintes. Les motifs finalement choisis sont indépendant au point de vue de prefix et suffix (prefix (suffix) d'un motif n'est également prefix(suffix) d'autres motifs). Ces motifs séquentiels raffinés sont des caractéristiques.
L'objectif de cet algorithme est le clustering des séquences de données en utilisant des motifs séquentiels et non le clustering des motifs séquentiels extraits eux-même. D'ailleurs l'efficacité de la méthode dépend du fait que les caractéristiques extraites doivent être présentes dans un nombre non-trivial de séquences de données. Ils ne proposent pas une mesure de similarité pour déterminer la similarité entre deux motifs séquentiels extraits. La complexité de la méthode de clustering est near-liner mais il faut considérer le coût de l'étape de sélection des caractéristiques et la projection des séquences de données dans l'espace vectoriel.

La méthode ApproxMAP [Kum04] est une combinaison du clustering et des motifs séquentiels pour arriver à l'extraction des alignement de motifs séquentiels multiples (multiple alignment sequential pattern mining). L'objectif de ApproxMAP est de chercher un motif séquentiel approximatif pour chaque cluster de séquences de données. Ce motif approximatif représente le contenu d'un cluster (ensemble de séquences). Notre objectif est de réaliser le clustering sur les motifs séquentiels extraits d'une base de séquences et donc de pouvoir comparer la distance entre deux motifs séquentiels. Or l'AproxMAP a pour but l'extraction d'une séquence approximative générale pour un cluster de séquences de données [Tan05].

3.4 Discussions

L'objectif est de proposer une mesure de similarité qui prend en compte toutes les caractéristiques de motifs séquentiels et aussi considère la sémantique de motifs séquentiels et les itemsets dans les motifs séquentiels. Les mesures présentées dans littérature présentent certains inconvénients dans le cas des motifs séquentiels ou elles ne sont appliquables que pour les séquences d'items. Le problème principal avec les mesures existant pour les séquences est le fait qu'elle traitent les motifs séquentiels comme une séquence d'éléments atomiques. Les éléments d'un motif séquentiel, c'est-à-dire les itemsets sont des ensembles d'items. Deux itemsets différents au niveau des items communs ne sont pas tout à fait deux événements différents.
Une mesure de similarité pour les motifs séquentiels, doit prendre en compte du fait que les motifs séquentiels sont des séquences ordonnées d'itemsets et non pas d'items. Elle doit également prendre en compte les positions (distance dans l'ordre) des itemsets lors du calcul de

la similarité. L'autre problème concerne le nombre des items non communs au niveau de la séquence et aussi au niveau des itemsets correspondant.

Propositions

N'allez pas où le chemin peut mener, allez là où il n'y a pas de chemin et laissez une trace.
EMERSON Ralph Waldo

Chapitre 4

Modélisation de comportements par clustering de motifs séquentiels

Dans le domaine de la détection d'intrusions, il existe deux approches principales :
(1) la détection d'anomalies,
(2) la détection d'abus (Misuse Detection).
Dans la première approche, les comportements normaux des utilisateurs du réseau sont connus et il est donc possible de construire des profils représentant ces comportements grâce à plusieurs caractéristiques comme les activités sur le réseau, etc. Une fois ces profils définis, nous comparons les nouvelles activités identifiées sur le réseau à ces profils et nous cherchons à détecter des déviations par rapport aux comportements normaux. Ces déviations peuvent alors correspondre à des intrusions. L'approche "Misuse Detection" est basée sur l'identification directe des attaques.

Notre proposition est une approche de la détection d'anomalies. Nous utilisons des motifs séquentiels fréquents pour modéliser les comportements. Les profils des comportements sont crées par clustering des motifs séquentiels fréquents extraits. Les intrusions sont identifiées en comparant les nouveaux comportements modélisés par les motifs représentant (motif central de chaque cluster) les profils. Dans la section suivante (c.f. section 4.1) nous introduisons notre approche et la section 4.1 est dédiée à l'explication détaillé de l'approche proposée.

4.1 Introduction de l'approche

Les systèmes de détection d'anomalies sont capables de trouver la plus part des intrusions même des intrusions inconnues. Cependant, ces systèmes génèrent un grand nombre des fausses alertes. Une anomalie est caractérisée comme une activité n'étant pas identifiée pendant les activités précédentes. Les anomalies trouvées sont considérées comme intrusions. La précision des système de détection d'anomalies dépend de la précision de la modélisation et de la flexibilité des comportements normaux.

Dans notre approche, nous modélisons deux types de comportements sur un réseau :

- les activités usuelles des utilisateurs du réseau,
- les intrusions multi-steps (les attaques multi-steps correspondent à plusieurs connexions consécutives).

Pour modéliser ces comportements, nous proposons d'utiliser des motifs séquentiels. La nature des motifs séquentiels, nous permet de modéliser ces comportements comme des évènements séquentiels fréquents dans le fil du temps. Les profils des comportements sont crées par clustering des motifs séquentiels fréquents extraits. Chaque cluster représente un groupe des comportements ressemblant. Dans la phase d'évaluation (*i.e.* la détection d'intrusions), les intrusions sont découvertes par la recherche des déviations éventuelles par rapport aux comportements normaux. Pour cela, nous comparons des nouveaux motifs séquentiels extraits des nouveaux logs (logs d'évaluation) avec des profils des comportements normaux. La figure 4.1 représente la démarche de notre approche.

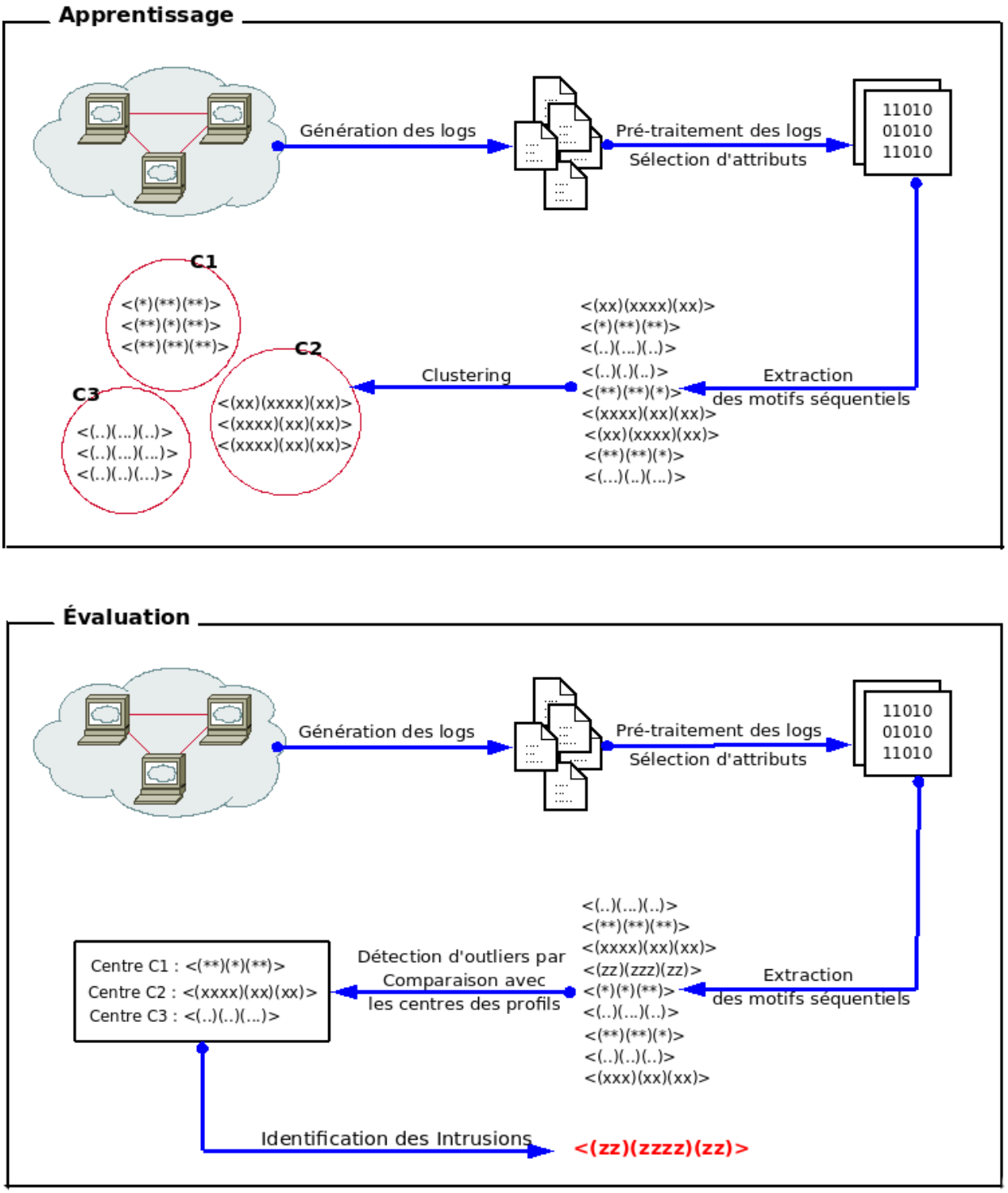

FIG. 4.1 – Les étapes de l'approche proposée pour la détection d'anomalies

Notre approche est capable de trouver les périodes temporelles dans lesquelles il y a des intrusions et les intrusions elles-mêmes. Cela dépend du format du jeu de données utilisé lors de l'extraction des motifs séquentiels et les solutions utilisées pour définir les notions de "client" et de "transaction".
Notons que cette proposition reste générale et appliquable sur d'autres domaines dans lesquels la modélisation des événement séquentiels ou la création des profils de comportements à partir des données séquentielles est une phase nécessaire. Dans la suite de cette proposition nous nous focalisons sur la détection d'anomalies.
L'approche proposée s'inscrit dans les méthodes classiques de détection d'intrusions mais que son originalité vient :

- de la modélisation des comportements normaux par le clustering des motifs séquentiels qui est adaptée pour décrire des évènements séquentiels comme les activités sur un réseau,
- de la définition d'une nouvelle mesure de similarité qui est adaptée aux motifs séquentiels (développée dans le chapitre 5).

  En effet, le clustering de motifs séquentiels pour la création des profils nécessite une mesure de similarité adaptée aux caractéristiques des motifs séquentiels. De même, la comparaison des motifs séquentiels lors de la phase d'évaluation pour trouver des déviations éventuelles ne peut s'effectuer que si une telle mesure est définie. Étant donné qu'il existe peu de travaux sur le clustering de motifs séquentiels et sur la mesure de similarité pour les motifs séquentiels et au vue des inconvénient des mesures existantes (cf. état de l'art - section 3), nous avons redéfini une mesure de similarité pour les motifs séquentiels. Nous développons cette mesure dans le chapitre 5.

4.2 Principe de l'approche

La méthode proposée afin de détecter les anomalies consiste en deux phases : (1) Une phase d'apprentissage qui consiste à identifier les comportements normaux des utilisateurs du réseau et à caractériser par le clustering de motifs séquentiels. (2) Une phase d'évaluation qui consiste à distinguer parmi les nouveaux comportements identifiés, ceux qui sont normaux et ceux qui sont anormaux.

4.2.1 Phase 1 - Apprentissage

Nous extrayons des motifs séquentiels sur des logs d'activités de réseau non-labelisés (c'est-à-dire non étiquetés comme normaux, anormaux ou liés à une attaque particulière). À partir de ces motifs, nous cherchons à construire des sous-ensembles de motifs qui se ressemblent. Pour effectuer ces regroupements, nous avons choisi d'utiliser la méthode de «clustering» puisque

nous ne connaissons pas les classes à priori dans lesquelles nous voulons classer les logs.
Une fois les clusters déterminés, nous choisissons l'un des motifs du cluster comme le centre du cluster grâce à la formule 4.1 pour obtenir le motif central :

$$Centre(C)_{C\in\{cluters\}} = \max_{Seq_i\in C}\{\sum_{j=0}^{n-1} Similarity\,(Seq_i, Seq_j)\} \tag{4.1}$$

1. **Apprentissage :**
   - **a.** Extraction de motifs séquentiels à partir de logs de réseau non-labelisés
   - **b.** Création des profils de comportements
     - **i.** Clustering des motifs extraits
     - **ii.** Recherche des séquences centrales de chaque cluster
2. **Évaluation (Détection d'intrusion)**
   - **a.** Extraction de nouveaux motifs séquentiels à partir des nouveaux logs
   - **b.** Identification des intrusions
     - **Première solution**
       - **i.** Clustering sur les nouveaux motifs
       - **ii.** Recherche de motifs centraux de chaque cluster
       - **iii.** Comparaison de la similarité des nouveaux motifs centraux avec ceux identifiés lors de la phase d'apprentissage
     - **Deuxième solution**
       - **i.** Comparaison de la similarité des nouveaux motifs avec les motifs centraux identifiés lors de la phase d'apprentissage

FIG. 4.2 – Approche proposée pour la détection d'anomalies

4.2.2 Phase 2 - Évaluation (Détection d'anomalies)

Nous allons extraire des motifs séquentiels à partir de logs de réseau. Nous cherchons ensuite à identifier parmi ces motifs ceux qui correspondent à des intrusions. Pour cela, nous avons envisagé deux solutions :

1. **Première solution :**
   Nous réalisons un clustering sur les nouveaux motifs séquentiels et cherchons les motifs séquentiels centraux de chaque cluster.

Nous comparons deux à deux le degré de similarité des motifs centraux des nouvelles séquences avec ceux de la phase d'apprentissage. Pour cela, nous utilisons une mesure de similarité adaptée aux caractéristiques particulières des motifs séquentiels que nous détaillons dans la section 5. Si un nouveau motif central ne ressemble à aucun motif central extrait lors de l'apprentissage (le degré de similarité est inférieur à un seuil défini), alors il peut s'agir d'une déviation par rapport aux comportements normaux.

Avantage En comparant seulement les séquences centrales, on diminue le nombre de comparaisons.

Inconvénient Il faut de nouveau réaliser le clustering lors de la phase d'évaluation sur les nouveaux motifs extraits.

2. **Deuxième solution :**
   Nous comparons deux à deux le degré de similarité des nouveaux motifs avec les motifs centraux déjà identifiés lors de l'apprentissage en utilisant une mesure de similarité dédiée aux motifs séquentiels.
   Si un motif séquentiel ne ressemble à aucun motif central (le degré de similarité est inférieur à un seuil défini), alors il peut s'agir d'une déviation par rapport aux comportements normaux.

   Avantage En comparant tous les nouveaux motifs aux motifs centraux, nous identifions des motifs qui ne ressemblent à aucun motif central et peuvent correspondre à des déviations.

   Inconvénient Cette solution nécessite un grande nombre de comparaisons de motifs.

La figure 4.2 reprend la structure de cette proposition.

Dans le chapitre 5, nous expliquons la mesure de similarité proposée pour les motifs séquentiels.

4.3 Discussions

La modélisation des comportements est une phase importante dans les IDS. Elle influence les résultats de l'IDS notamment au niveau du taux de fausses alertes. Les comportements des utilisateurs d'un réseau sont des activités séquentielles. Les comportements usuels sont normalement répétés au fils du temps. Nous avons proposé une approche de la détection d'anomalies fondée sur le clustering de motifs séquentiels. Les motifs séquentiels nous permettent de trouver des corrélations entre des événements séquentiels au fils du temps. C'est la raison pour la quelle nous cherchons les comportements séquentiels fréquents à l'aide des motifs séquentiels. Le clustering des motifs séquentiels regroupe les activités similaires et nous aide de créer des profils de

comportements. Chaque profil est représenté par un motif séquentiel correspondant au centre du cluster associé à profil. Dans la phase d'évaluation, on compare la similarité des nouveaux motifs extraits avec les centres des profils. Les motifs séquentiels qui ne sont proches d'aucun centre sont identifiés comme une déviation aux comportements normaux et donc comme une intrusion. Notre approche est capable de trouver les intervalles temporels contenant les intrusions ainsi que les activités intrusives. Cela dépend du format de l'extraction des motifs séquentiels (*i.e.* la définition des notions de "client" et de "transaction").

Chapitre 5

Une mesure de similarité pour les Motifs séquentiels

Dans notre approche de détection d'anomalies, le clustering de motifs séquentiels pour la création des profils ainsi que la comparaison des motifs séquentiels lors de la phase d'évaluation pour trouver des déviations éventuelles, nécessitent une mesure de similarité adaptée aux caractéristiques des motifs séquentiels.
D'après la littérature, il y a peu de recherche portant sur le clustering de motifs séquentiels ou sur la définition d'une mesure de similarité dédiée aux motifs séquentiels. Les travaux existant dans cette domaine sont bien expliqués et analysés dans le partie l'état de l'art, section 3.
Nous avons défini une mesure de similarité pour la comparaison des motifs séquentiels qui prend en compte les caractéristiques particulières des motifs séquentiels. Dans notre contexte, cette mesure est utilisée pour faire du clustering sur les motifs séquentiels extraits et trouver des déviations par rapport aux comportements normaux. Par contre, notre mesure de similarité peut être utile dans autres domaines où il faut comparer des motifs séquentiels. Par exemple dans l'extraction des motifs séquentiels sous contrainte de similarité, la visualisation des motifs séquentiels et la compression des motifs séquentiels. Cette mesure est bien adapté aux cas il y a des comparaisons directionnelles. C'est à dire lorsque l'on compare des motifs séquentiels avec un motif de référence comme l'extraction des motifs séquentiels sous contrainte ou la visualisation. Dans la section suivante(5.1) nous détaillons et formalisons notre mesure.

5.1 Description générale

Cette mesure est composée de trois scores : (1) le score de mapping qui mesure la ressemblance de deux motifs en fonction des liens qu'il est possible d'établir entre les itemsets, (2) le score d'ordre qui mesure la ressemblance des deux séquences vis-vis de l'ordre des itemsets.

Nous détaillons le calcul de ces deux scores, (3) la dernière phase est le calcul du degré de similarité en fonction du score de mapping et de celui d'ordre.

Phase 1 – Calcul du score de mapping :
Nous cherchons pour chaque itemset i de la séquence 1 $Seq_1(i)$, l'itemset j le plus ressemblant dans la séquence 2 $Seq_2(j)$. Ensuite, nous faisons correspondre ces deux itemsets et nous donnons à chaque couple de mapping un poids qui est le degré de similarité des deux itemsets. Pour calculer cette similarité des deux itemsets, nous calculons le nombre normalisé (divisé par la moyenne du nombre total d'items de deux itemsets)des items communs entre deux itemsets [GK01].

Définition 5.1. $Poids(i)(j)$ entre le i'ème itemset de la Seq_1 et le j'ème itemset de la Seq_2 :

$$poids(i)(j) = \frac{\mid Seq_1(i) \cap Seq_2(j) \mid}{\frac{|Seq_1(i)|+|Seq_2(j)|}{2}} \tag{5.1}$$

Formule 5.1– Le poids du mapping de deux itemsets

Définition 5.2. Soit $Seq_1(i)$ le i'ème itemset de la Seq_1 et $Seq_1(i)$ le j'ème itemset de la Seq_2, le **mapping des itemsets** se fait ainsi :

$$Mapping(Seq_1(i), Seq_2(j)) \mid poids(i)(j) = \max_{x,y=0}^{n}(poids(x)(y)) \wedge poids(i)(j) \neq 0 \tag{5.2}$$

Formule 5.2– Le mapping de deux itemsets

Un itemset de la Seq_2 dont le poids est le plus élevé sera choisi pour le mapper avec l'itemset courant de la Seq_1.

Principe 1 Pour les poids égaux, nous prenons l'itemset ayant un $timeStampe$ (ts)[1] inférieur, c'est-à-dire l'itemset qui est situé avant les autres dans l'ordre de la séquence afin de respecter l'ordre des itemSets.

$$Mapping(Seq_1(i), Seq_2(j)) \mid poids(i)(j) = poids(i)(k) = \max_{x,y=0}^{n}(poids(x)(y)) \wedge ts(j) < ts(k) \wedge poids(i)(j) \neq 0 \tag{5.3}$$

Formule 5.3– Le mapping lors l'existence des candidats du mapping ayant poids égaux

Nous avons formalisé la démarche détaillée de la phase du mapping dans l'algorithme 1.

[1] Le $timeStampe$ dans notre contexte, correspond à la position des itemsets dans l'ordre de la séquence. Par exemple dans la séquence (a)(ab)(c), le $ts(ab) = 2$.

Algorithme 1 : Mesure de la similarité, Phase 1 : Mapping des itemSets ressemblants

Data : $Seq_1 = \langle Seq_1(1), ..., Seq_1(n)\rangle$; $Seq_2 = \langle Seq_2(1), ..., Seq_2(m)\rangle$
Result : $MapOrder$: Un tableau qui garde les itemsets associés de la Seq_2 dans l'ordre des itemSets de la Seq_1 ; $MapOrder(l)$ = l'itemset associés à $l'em$ itemset de la Seq_1

for $i \leftarrow 1$ **to** n **do**
  take $Seq_1(i)$;
  $exclusItemSets \longleftarrow \{\}$;
  for $j \leftarrow 1$ **to** m **do**
    take $Seq_2(j)$;
    $Poids(i)(j) = \frac{Seq_1(i) \cap Seq_2(j)}{\frac{|Seq_1(i)|+|Seq_2(j)|}{2}}$;
  $mapCandidat \longleftarrow$ `TakeMostWeightedItemSet(`$Poids, i$`)`;
  if $mapCandidat \in \{MappedItemSets\}$ **then**
    $Seq_1(k) \longleftarrow MappedItemSets(mapCandidat)$;
    $newCandidat_1$, $newCandidat_2 \longleftarrow$ `SolveConflict(`*mapCandidat,* $Seq_1(i)$*,* $Seq_1(k)$*,* *exclusItemSets*`)`;
    repeat
      if $newCandidat_1 = mapCandidat$ **then**
        $MapOrder(i) \longleftarrow mapCandidat$;
        $MappingWeigh(mapCandidat) = Poids(Seq_1(i), mapCandidat)$;
        if $newCandidat_2 \in \{MappedItemSets\}$ **then**
          $Seq_1(k) \longleftarrow MappedItemSets(newCandidat_2)$;
          $newCandidat_1$, $newCandidat_2 \longleftarrow$ `SolveConflict(`$newCandidat_2$*,i,k,exclusItemSets*`)`;
          Continue next Repeat;
        else
          $MapOrder(k) \longleftarrow newCandidat_2$;
          $MappingWeigh(newCandidat_2) = Poids(Seq_1(k), newCandidat_2)$;;
          $flag = false$;
      else
        $MapOrder(k) \longleftarrow mapCandidat$;
        $MappingWeigh(mapCandidat) = Poids(Seq_1(k), mapCandidat)$;
        if $newCandidat_1 \in \{MappedItemSets\}$ **then**
          $Seq_1(k) \longleftarrow MappedItemSets(newCandidat_1)$;
          $newCandidat_1$, $newCandidat_2 \longleftarrow$ `SolveConflict(`$newCandidat_1$*,i,k,exclusItemSets*`)`;
          Continue next Repeat;
        else
          $MapOrder(i) \longleftarrow newCandidat_1$;
          $MappingWeigh(newCandidat_1) = Poids(Seq_1(i), newCandidat_1)$;
          $flag = false$;
    until *flag* ;
  else
    $MapOrder(i) \longleftarrow mapCandidat$;
    $MappingWeigh(mapCandidat) = Poids(Seq_1(i), mapCandidat)$;
Return MapOrder;

Cas particulier : conflit de mapping Un itemset sélectionné de la 2ème séquence pour le faire correspondre à un itemset de la séquence 1 peut avoir déjà été mappé avec un autre itemset de la séquence 1. Nous appelons cette situation "conflit de mapping". Pour résoudre ce problème, il fait trouver un autre candidat de mapping pour l'un des itemsets en conflit. Pour cela, nous utilisons la fonction $SolveConflict$ qui propose un nouvel itemset non déjà mappé. Nous allons détailler la fonction $SolveConflict$ dans le principe 2 :

Principe 2 Si l'itemset sélectionne $Seq_2(j)$ dans la 2ème séquence pour le mapping avec l'itemset $Seq_1(i)$ de la 1ère séquence est déjà associé à un autre itemset $Seq_1(k)$ de la 1ère séquence , nous appelons la fonction :
$SolveConflict(mapCandidat; Seq_1(i); Seq_1(k); exclusItemSets)$

Les entrées de la fonction $SolveConflict$ sont respectivement :

- $mapCandidat = Seq_2(j)$ comme le candidat actuel pour le mapping avec l'itemset courant $Seq_1(i)$
- les deux itemsets $Seq_1(i)$, $Seq_1(k)$ qui sont en conflit : $Seq_1(i)$ est l'itemset courant et $Seq_1(k)$ est l'itemset qui est déjà mappé à l'itemset $Seq_2(j)$ ($mapCandidat$).
- $exclusItemSets$ = un ensemble d'itemsets exclus de la 2ème séquence. Cet ensemble contient les itemsets ayant déjà été proposés pour résoudre le conflit actuel. En effet, c'est possible que un nouveau candidat proposé pour résoudre de conflit soit aussi déjà mappé. Donc nous l'excluons lors de la prochaine recherche de candidat.

Dans la fonction $SolveConflict$ nous calculons les similarités locales pour tous les candidat du mapping. Ainsi, pour chacun des itemsets $Seq_1(i)$ et $Seq_1(k)$ en conflit, nous cherchons deux autres candidats dans la Seq_2 pour le mapping (autre que $Seq_2(j)$, l'itemset candidat actuel) :

- Le 1er candidat est l'itemset se situant avant $Seq_2(j)$ et qui possède également le poids maximum parmi les itemsets placés avant $Seq_2(j)$.
- Le 2ème candidat est sélectionné de la même manière mais il est recherché après $Seq_2(j)$.

Ces nouveaux candidats pour l'itemSet $Seq_1(i)$ et l'itemSet $Seq_1(k)$ sont appellés $nextMaxAfter_i$, $nextMaxBefor_i$ et respectivement $nextMaxAfter_k$, $nextMaxAfter_k$.

Nous créons ensuite tous les couples possibles de mappings. Pour cela, en associant : $Seq_2(j)$ à $Seq_1(i)$ on mappe une fois $nextMaxBefore_k$ avec $Seq_1(k)$ et une fois $nextMaxAfter_k$ avec $Seq_1(k)$ puis on procède de même en associant cette fois $Seq_2(j)$ à $Seq_1(k)$ et en mappant

une fois $nextMaxBefore_i$ avec $Seq_1(i)$ et une fois $nextMaxAfter_i$ avec $Seq_1(i)$. Finalement nous obtenons les couples du mapping suivant :

- $< Seq_1(i), Seq_2(j) >, < Seq_1(k), nextMaxBefore_k >$
- $< Seq_1(i), Seq_2(j) >, < Seq_1(k), nextMaxAfter_k >$

- $< Seq_1(k), Seq_2(j) >, < Seq_1(i), nextMaxBefore_i >$
- $< Seq_1(k), Seq_2(j) >, < Seq_1(i), nextMaxAftre_i >$

Pour ces couples, nous calculons un score appelé $localSim$. Ce score permet d'évaluer si le couple correspondant est pertinent comme nouveaux candidats du Mappping. Il s'agit d'une similarité partielle (*locale*) entre deux motifs séquentiels. Le calcul de $localSim$ se fait ainsi : – **Si** $X_1 \equiv Seq_1(k)$ & timeStampe(Can_2)>timeStampes(Can_1) || $X_1 \equiv Seq_1(i)$ & timeStampe(Can_2)<timeStampes(Can_1)

$$localSim(X_1, Can_1)(X_2, Can_2) = \frac{Poids((X_1, Can_1)) + Poids((X_2, Can_2))}{2} \quad (5.4)$$

– **Sinon**

$$localSim(X_1, Can_1)(X_2, Can_2) = \frac{1}{2} \times \frac{Poids((X_1, Can_1)) + Poids((X_2, Can_2))}{2} \quad (5.5)$$

La condition permet de vérifier si les itemsets mappés de la Seq_2 sont dans la même ordre comme ceux de la Seq_1. Quand l'ordre n'est pas respecté lors du mapping (mapping croisé, cf. Fig 5.1), le $localsim$ est divisé par 2. En effet, cela veut dire que seulement un itemset parmi les deux itemsets de la 2ème séquence (e.g. Sur la Fig 5.1, l'un parmi *(can1)* et *(cand2)*) peut être considéré dans l'ordre des deux itemsets de la 1ère séquence (les deux itemsets *(X1)* puis *(X2)*). L'ordre étant à moitié respecté, on divise par deux le score $localSim$ (lors de mapping croisé). Cette division peut être aussi justifiée par les formules (5.6),(5.7) et (5.8) de la phase de calcul du score de l'ordre.

En choisissant le couple ayant le $localSim$ le plus élevé, nous sélectionnons les nouveaux candidats pour les mapper avec les itemsets $Seq_1(i)$ et $Seq_1(k)$. Notons que à la fin, $mapCandidat$ (*i.e.* le candidat initial ou autrement dit $Seq_2(j)$) est proposé soit comme un candidat pour $Seq_1(i)$, soit comme un candidat pour $Seq_1(k)$ en fonction de la valeur du $localSim$. L'agorithme 2 correspond à cette procédure de résolution de conflits.

Cas particulier : boucle de conflit L'algorithme 1 gère le cas où il y a un boucle de conflit (*i.e.* lorsque le candidat renvoyé par la fonction $solveConflict$ est aussi mappé avec un autre itemset). Quand l'itemset candidat suivant (le sortie de la fonction $SolveConflict$) qui est sélectionné pour résoudre de conflit est lui-même mappé à autre itemset, nous continuons à

FIG. 5.1 – Mapping croisé et ordre des itemsets

faire appel à la fonction $SolveConflict$ jusqu'à ce qu'il n'y ait plus de conflit.

Phase 2 – Calcul du score de l'ordre :
L'objectif du score de l'ordre est de trouver les mappings qui ne sont pas croisés en même temps qu'il détermine la distance (dans l'ordre) entre les couples mappées. Cela veut dire que, nous cherchons à savoir si les couples de mapping sont consécutifs ou il existe des espaces (les itemsets non mappés) entre les couples. La figure 5.2 montre un exemple de mapping consécutif et aussi un exemple de mapping non consécutif. Une fois le mapping réalisé pour

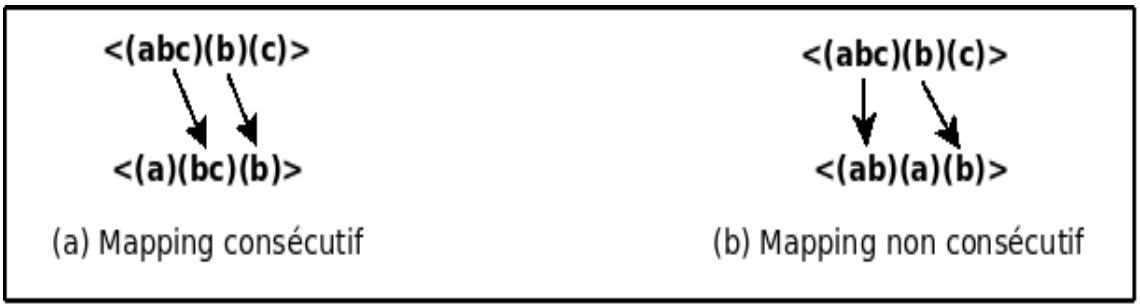

FIG. 5.2 – Mapping consécutifs et non consécutifs en fonction d'ordre des itemsets

tous les itemsets, nous cherchons le pourcentage des itemsets mappés de séquence 2 étant dans la même ordre que des itemsets de la séquence 1. Pour cela nous réordonnons d'abord les itemsets de la séquence 2 dans l'ordre du mapping avec les itemsets de la séquence 1 et puis en récupérant les timeStampes des itemsets de la séquence 2, nous créons une liste nommée $mapOrder$. Autrement dit pour créer la liste $mapOrder$, nous mettons à la 1ère place dans la liste le timeStampe de l'itemset mappé au 1ère l'itemset de la séquence 1. En 2ème place, le timeStampe de l'itemset mappé avec le 2eme itemset de la séquence 1 et ainsi de suite.

Algorithme 2 : Résolution du conflit lors du Mapping des itemsets ressemblants

Data : **mapCandidat,** $Seq_1(i)$, $Seq_1(k)$, **exclusItemSets** : Un itemset candidat de la Seq_2 et deux itemSets $Seq_1(i)$, $Seq_1(k)$; $Seq_1(i)$ est l'itemset courant
Data : **Poids** : Un tableau contenant les poids de comparaison des itemsets
Result : $newCandidat_1$, $newCandidat_2$: $newCandidat_1$ pour associer à $Seq_1(i)$ et $newCandidat$ pour associer à $Seq_1(k)$

$nextMaxAfter_i \longleftarrow$ `NextMostWeightedItemSet(`$Seq_1(i)$, *after*, $Seq_2(j)$`)`;
$nextMaxBefor_i \longleftarrow$ `NextMostWeightedItemSet(`$Seq_1(i)$, *befor*, $Seq_2(j)$`)`;
$nextMaxAfter_k \longleftarrow$ `NextMostWeightedItemSet(`$Seq_1(k)$, *after*, $Seq_2(j)$`)`;
$nextMaxBefor_k \longleftarrow$ `NextMostWeightedItemSet(`$Seq_1(k)$, *befor*, $Seq_2(j)$`)`;

$nextMaxs_{Seq_1(i)} \longleftarrow \{nextMaxAfter_i, nextMaxBefor_i\}$
$nextMaxs_{Seq_1(k)} \longleftarrow \{nextMaxAfter_k, nextMaxBefor_k\}$

$MapTuple \longleftarrow ((A,B),(C,D))$
$A \in \{Seq_1(i), Seq_1(k)\}$
$B \in \{Seq_2(j)\}$
$C \in \{Seq_1(i), Seq_1(k)\} \vee C \neq A$
$D \in \boldsymbol{nextMaxs_x} \vee x = C \vee D \neq \emptyset$

forall *Possible cases of MapTuple* **do**
  if *(*$A \equiv Seq_1(k)$ *AND timeStampe(D) > timeStampes(*$Seq_2(j)$*)) OR*
  ($A \equiv Seq_1(i)$ *AND timeStampe(D) < timeStampes(*$Seq_2(j)$*))* **then**
    $localSim(A,B)(C,D) = \dfrac{Poids((A,B)) + Poids((C,D))}{2}$;
  else
    $localSim(A,B)(C,D) = \dfrac{1}{2} \times \dfrac{Poids((A,B)) + Poids((C,D))}{2}$;

$maxLocalSim(A,B)(C,D) \longleftarrow \max\{all\ possibles\ localSim(A,B)(C,D)\}$;

if *in maxLocalSim(A,B)(C,D)* $A \equiv Seq_1(i)$ **then**
  Return $newCandidat_1 \longleftarrow$ B in $maxlocalSim(A,B)(C,D)$;
  Return $newCandidat_2 \longleftarrow$ D in $maxlocalSim(A,B)(C,D)$;
else `/* A` $\equiv Seq_1(k)$ `*/`
  Return $newCandidat_2 \longleftarrow$ B in $maxlocalSim(A,B)(C,D)$;
  Return $newCandidat_1 \longleftarrow$ D in $maxlocalSim(A,B)(C,D)$;

Finalement $mapOrder$ forme un liste des entiers (timeStamps) ainsi :

$$mapOrder = \{t_1, t_2, ..., t_i, ..., t_n\}$$

t_i = le timeStamp de l'itemset de la Seq_2 associés à i'ème itemSet de la Seq_1.

Ensuite nous cherchons les sous-séquences croissantes et optimales de la liste $MapOrder$. Pour chacune nous calculons deux scores :
(1) $totalOrder$,
(2) $ConsecutiveOrder$.
$totalOrder$ mesure quel pourcentage du $mapOrder$ à été apparaît dans la sous-séquence croissante. En considérant le concept de la liste $mapOrder$, le score $totalOrder$ montre la proportion de l'ordre de mapping qui a été fait dans l'ordre des itemsets de la séquence 1.
$ConsecutiveOrder$ donne un score par rapport aux gaps éventuels entre les éléments de la sous-séquence croissante ou autrement dit le pourcentage des timeStamps exactement consécutifs dans la sous-séquence.

$$totalOrder = \frac{nbOrderedItemSets}{nbAllItemSets} \tag{5.6}$$

nbAllItemSets = nombre total des itemsets

nbOrderedItemSets = nombre des itemsets dans la sous-séquence ordonnée

$$consecutiveOrder = \frac{nbOrderedItemSets}{timeStampe(lastItemSet) - timeStampe(firstItemSet) + 1} \tag{5.7}$$

lastItemSet = dernier itemset dans la sous-séquence ordonnées
firstItemSet = 1er itemset dans la sous-séquence ordonnées

Le score de l'ordre final $orderMeasure$ pour chaque sous-séquence croissante correspond à la multiplication du score $totalOrder$ et du score $ConsecutiveOrder$.

$$orderMeasure = totalOrder \times consecutiveOrder \tag{5.8}$$

Pourquoi calculer toutes les sous-séquences croissantes et non seulement la sous-séquence croissante la plus longue ? À première vue, chercher la sous-séquence la plus longue nous donne la serie du mapping la plus longue où les mapping ne sont pas croisés. Or, une analyse approfondie nous montre que chercher la sous-séquence croissante la plus longue (*i.e.* la série du mapping la plus longue) n'est toujours pas pertinente. Nous l'expliquons par l'exemple suivant :

Prenons les deux mappings "1" et "2" sur la figure 5.3, la série de mapping "1" est plus petit

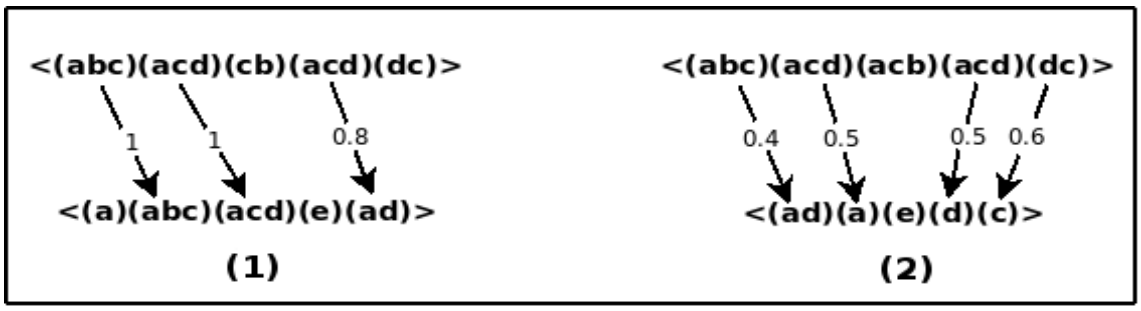

FIG. 5.3 – Mapping et l'ordre des itemsets

que la série de mapping "2" (sur la figure 5.3-1, ils existent 3 mappings et sur la figure 5.3-2 il en exite 4). Les deux séquences sur la figure 5.3-1 sont plus proches que les deux séquences sur la figure 5.3-2. Par contre, si on se contante de la série la plus longue (*i.e.* figure 5.3-2), on perd l'influence des poids des mappings sur le degré de similarité final. Par exemple ici, la moyenne des poids des mapping sur la figure 5.3-1 est égale à 0.93 et sur la figure 5.3-2 égale à 0.5. Par conséquent la meilleure solution est de considérer toutes les séries du mapping possibles (toutes les sous-séquences croissantes) et finalement choisir celle qui possède le score maximum, c-à-d. celle qui satisfait la formule 5.9.

Phase 3 – Calcul de la similarité :

Pour chaque sous-séquence croissante trouvée, nous calculons le score de similarité $simScore$ par une agrégation entre le score de l'ordre $orderMeasure$ et la moyenne des poids du Mapping $AveWeightScore$ des itemsets apparaissant dans la sous-séquence. Le calcul de la moyenne des poids permet de prendre en compte l'influence du poids de chaque itemset sur la totalité de la séquence. Ainsi, nous prenons en compte les effets des mapping ayant un $Poids$ égal à 0 (il n'y a aucun itemset ressemblant pour l'itemset correspondants). Avec la moyenne des Poids $AveWeightScore$, nous prenons en compte l'importance de chaque itemset.

Le $simScore$ le plus élevé sera le degré de similarité $patternsSimDegree$ entre les deux motifs

séquentiels. L'algorithme 3 formalisme les phase 2 et 3.

$$patternsSimDegree = \max\{orderMeasure(sub) \times AveWeightScore(sub)\} \quad (5.9)$$

$$sub \in \{sous\_seqs\, croissantes\, et\, optimales\, du\, mapOrder\}$$

Algorithme 3 : Mesure de la similarité, Phases 2 et 3

Data : **MapOrder** : Tableau retourné de la phase 1 contenant des itemSets associés de la Seq_2 et ordonnés par rapport de l'ordre des itemSets de la Seq_1
Data : **MappingWeight** : Un Tableau Associatif retourné de la phase 1 pour garder les poids du Mapping, Key :$Seq_2(j)$ et Value :$Poids(*)(j)$
Result : **SimilarityDegree** : Degré de similarité entre deux motifs séquentiels

foreach *ordered sub-sequence of MapOrder* **do**

```
                                /* ordonnée par rapport aux timeStampes des itemSets */
```

$totalOrder \longleftarrow \frac{nbOrderedItemSets}{nbAllItemSets}$;

```
                                /* nbAllItemSets = nombre des itemSets dans MapOrder */
          /* nbOrderedItemSets = nombre des itemSets dans la sous-séquence ordonnées */
```

$ConsecutiveOrder \longleftarrow \frac{nbOrderedItemSets}{timeStampe(lastItemSet)-timeStampe(firstItemSet)+1}$;

$orderMesure \longleftarrow totalOrder \times ConsecutiveOrder$;

$AveWeightScore \longleftarrow \frac{\sum_{\forall itemSet \in sub\_seq} MappingWeight(itemSet)}{|sub\_sequence|}$;

$SimilarityDegree \longleftarrow orderMesure \times AveWeightScore$;

$MaxSimilarityDegree \longleftarrow \max\{SimilarityDegree\}$;

Return $MaxSimilarityDegree$;

5.1.1 Exemple détaillé de calcul de la mesure de similarité

Le calcul de la similarité est expliqué dans l'exemple suivant. Nous avons essayé de considérer la plupart des cas ainsi que les conflits éventuellement rencontrés avec notre approche.
Soit deux motifs séquentiels : $Seq_1 = \{(abc)(ab)(cd)\}$ et $Seq_2 = \{(ab)(ca)(a)\}$ pour lesquels nous souhaitons calculer la similarité.

Phase 1 : Calcul du score de mapping

Pour chaque itemset $Seq_1(i)$ de la 1ère Séquence, on cherche l'itemset $Seq_2(j)$ le plus ressemblant dans la 2éme séquence. Pour cela, nous comparons deux itemsets en utilisant la formule

(5.1).

- $Poids\,(Seq_1(1), Seq_2(1)) \Longrightarrow Poids\,((abc),(ab)) = \frac{2}{\frac{3+2}{2}} = 0.8$
- $Poids\,(Seq_1(1), Seq_2(2)) \Longrightarrow Poids\,((abc),(ca)) = \frac{2}{\frac{3+2}{2}} = 0.8$
- $Poids\,(Seq_1(1), Seq_2(3)) \Longrightarrow Poids((abc),((a)) = \frac{1}{\frac{3+1}{2}} = 0.5$

Les deux premiers $Poids$ sont égaux, donc d'après le **principe 1** on sélectionne le 1er itemsets de la séquence 2 (*i.e.* (ab)) pour associé à l'itemsets $Seq_1(1)$ (*i.e* (abc)). On a donc : $MappedItemSets.put(Seq_1(1), Seq_2(1))$.

Nous continuons par l'itemset suivant $Seq_1(2)$ de la séquence un :

- $Poids(Seq_1(2), Seq_2(1)) \Longrightarrow Poids((ab),(ab)) = 1$
- $Poids(Seq_1(2), Seq_2(2)) \Longrightarrow Poids((ab),(ca)) = 0.5$
- $Poids(Seq_1(2), Seq_2(3)) \Longrightarrow Poids((ab)(a)) = 0.6$

D'après le calcul nous devrons sélectionner l'itemset $Seq_2(1)$. Or cet itemset $Seq_2(1)$ à déjà été associé à l'itemset $Seq_1(1)$. Par conséquent, nous utiliserons la fonction de résoudre de conflit $SolveConflict(mapCandidat; Seq_1(i); Seq_1(k); exclusItemSets)$ avec $mapCandidat = Seq_2(1)$, $Seq_1(i) = Seq_1(2)$, $Seq_1(k) = Seq_1(1)$, $exclusItemSets = \phi$. Avec cet l'algorithme, nous cherchons de nouveaux candidats pour les itemsets en conflit $Seq_1(1)$, $Seq_1(2)$ avant et après l'itemset candidat actuel $Seq_2(1)$. Nous obtenons les candidats suivant :

- **pour l'itemset $Seq_1(1)$:**
  $nextMaxBefore_1 = \emptyset$
  $nextMaxAfter_1 = Seq_2(2)$
- **pour l'itemset $Seq_1(2)$:**
  $nextMaxBefore_2 = \emptyset$
  $nextMaxAfter_2 = Seq_2(3)$

- **Les couples Mappings possibles :**
  $((Seq_1(1), Seq_2(1)), (Seq_1(2), Seq_2(3)))$
  $((Seq_1(2), Seq_2(1)), (Seq_1(1), Seq_2(2)))$.

En respectant la condition lors du calcul de $localSim$ expliqué dans les formules(5.4) et(5.5) et en utilisant les poids du mapping, nous obtenons :

$localSim((Seq_1(1), Seq_2(1)), (Seq_1(2), Seq_2(3))) = \frac{0.8+0.6}{2} = 0.7$
$localSim((Seq_1(2), Seq_2(1)), (Seq_1(1), Seq_2(2))) = \frac{1}{2} \times \frac{1+0.8}{2} = 0.45$

Nous choisissons le couple ayant le $localSim$ le plus élevé : $\langle(Seq_1(1),\ Seq_2(1))$; $(Seq_1(2),\ Seq_2(3))\rangle$. L'itemset $Seq_2(1)$ est associé avec $Seq_1(1)$ et l'itemset $Seq_2(3)$ avec $Seq_1(2)$. Après application du **principe 2** et vérification des conditions, on a comme Mapping :
$MappedItemSets.put(Seq_1(2), Seq_2(3))$
$MappedItemSets.put(Seq_1(1), Seq_2(1))$

Nous procédons de la même manière pour le 3ème itemset $Seq_1(3)$ de la 1ère séquence et nous obtenons : $MappedItemSets.put(Seq_1(3), Seq_2(2))$.

A la fin de l'etape de Mapping, les Mappings sont :

Mapping($Seq_1(1) = (abc)$, $Seq_2(1) = (ab)$)
Mapping($Seq_1(2) = (ab)$, $Seq_2(3) = (a)$)
Mapping($Seq_1(3) = (cd)$, $Seq_2(2) = (ca)$)

Phase 2 – Calcul du score de l'ordre :
Il s'agit de comparer l'ordre des itemsets dans les deux séquences. Nous réordonnons les itemsets de la 2ème séquence par rapport à l'ordre du Mapping avec les itemsets de la 1ere séquence pour créer la liste $mapOrder$.Pour cela, comme déjà expliqué, nous mettons pour la i'ème place dans $mapOrder$ le timeStampe de l'itemset mappé au i'ème itemset $Seq_1(i)$ de la 1ère séquence. Donc, le 1ère place dans la liste $mapOrder$ est prie par "1" d'après le timeStampe de l'itemset $Seq_2(1)$ étant mappé avec le 1er itemset $Seq_1(1)$ de la 1ère séquence. Pour la 2ème place, le timeStampe de l'itemset $Seq_2(3)$ mappé avec le 2ème itemset $Seq_1(2)$ de la 1ère séquence (*i.e.* "3") et dans la même manière nous mettons "2" dans la 3ème place dans la liste $mapOrder$. Par conséquent le $mapOrder$ est :

$mapOrder = 1, 3, 2$

Nous cherchons les sous-séquence croissante et optimale de la séquence $mapOrder$.

- **Les deux sous-séquences croissantes optimales trouvées** :
  - *sous-seq*$_1 = (1, 2)$
  - *sous-seq*$_2 = (1, 3)$
- **D'après les formules (5.6),(5.7) et (5.8), le score de l'ordre est :**
  - orderMeasure((1,2)) = $\frac{2}{3} \times \frac{2}{2-1+1} = \frac{2}{3}$
  - orderMeasure((1,3)) = $\frac{2}{3} \times \frac{2}{3-1+1} = \frac{4}{9}$

Phase 3 – Calcul de la similarité

Nous ajoutons maintenant la moyenne des poids $AveWeightScore$ des itemsets apparaissant pour chaque sous-séquences croissante. Alors pour les sous-séquences, nous calculons la moyenne des poids des itemsets :

sous-seq(1,2) : $(Seq_2(1),\ Seq_2(2)) = \dfrac{0.8 + 0.5}{2} = 0.65$
sous-seq(1,3) : $(Seq_2(1),\ Seq_2(3)) = \dfrac{0.8 + 0.6}{2} = 0.7$

Avec une multiplication entre la mesure d'ordre et la moyenne des poids et puis en sélectionnant les cas ayant la valeur la plus élevée, nous obtenons le degré de similarité entre les deux motifs séquentiels :

sous-seq(1,2) : $(Seq_2(1),\ Seq_2(2)) = \frac{2}{3} \times 0.65 = 43\%$
sous-seq(1,3) : $(Seq_2(1),\ Seq_2(3)) = \frac{4}{9} \times 0.7 = 31\%$

Le degré de similarité $patternsSimDegree$ entre les deux motifs séquentiels de l'exemple est égal à 43%.

$patternsSimDegree = \max\{43, 31\} = 43\%$

5.2 Discussions

Afin de comparer la similarité des motifs séquentiels, nous avons défini une mesure de similarité qui prend en compte les caractéristiques et la sémantique des motifs séquentiels. Cette mesure compare les deux motifs séquentiels à la fois au niveau des itemsets et leurs positions dans la séquence et aussi au niveau des items dans les itemsets ressemblant. Notre mesure est

combinée de deux scores : (1) score donné aux poids du mapping des itemsets (2) score donné selon la ressemblance des itemsets correspondant au niveau de leurs positions dans les deux séquences. Ces deux scores nous permettent de surmonter les problèmes liés aux mesures LCS et Edit distance dans le domaine des motifs séquentiels (cf. section 3.2).

Cette mesure est a priori une mesure **asymétrique**. Elle est bien adapté aux applications dans lesquelles la comparaison est directionnelle. Dans le domaine de l'extraction des motifs séquentiels sous contrainte de similarité, la visualisation des motifs proches d'un motif sélectionné ou plus généralement lorsqu'il y a de comparaisons des motifs séquentiels avec un motif de référence, une mesure de similarité asymétrique est bien adaptée. Dans les cas où on a besoin d'une mesure symétrique comme le clustering, nous rendons notre mesure symétrique en calculant la moyenne des degrés de similarité dans les deux directions. Nous notons qu'il est possible de convertir cette mesure en une mesure symétrique en modifiant les principes du mapping. Cela ne change pas la sémantique de l'approche mais la complexité de l'algorithme. Cette étude est un perspective de ce projet. Nous allons implémenter la version symétrique de notre mesure et la comparer avec la version asymétrique au niveau de la complexité.

Expérimentations

In theory, there is no difference between theory and practice. But, in practice, there is.

Jan L.A. van de Snepscheut

Chapitre 6

Expérimentations

Dans le cadre ce projet, nous avons introduit une mesure de similarité pour les motifs séquentiels. Cette mesure de similarité peut servir dans le cadre du clustering de motifs séquentiels, l'extraction des motifs séquentiels sous contrainte de similarité, la compression de motifs séquentiels, la visualisation des motifs séquentiels ressemblant etc. Dans toutes ces applications, la comparaison de la similarité est une phase interne d'un autre algorithme comme clustering ou l'extraction sous contrainte. C'est pourquoi la mesure de similarité doit être rapide au niveau du temps de calcul. Malgré l'apparence complexe de l'algorithme de notre mesure, nous allons montrer que notre mesure de similarité est très efficace au niveau du temps d'exécution et de la taille de mémoire utilisée. Cette mesure est donc utilisable au sein de n'importe quelle autre application sans la ralentir. Pour cela, nous définissons un protocole d'expérimentation. Au sein de ces expérimentations, les trois facteurs influencant le temps d'exécution et la taille de mémoire utilisée sont considérés. En effet, notre mesure a résisté aux tests menés sur les deux critéres.

6.1 Protocole d'expérimentation

Nous testons le temps d'exécution et la taille de mémoire utilisée de notre mesure de similarité dans trois directions. (1) Nous cherchons le changement du temps d'exécution lorsque le nombre des itemsets augmente. (2) Nous nous intéressons au changement du temps quand le nombre d'items (*i.e.* la longueur) dans les motifs séquentiels augmente et finalement (3) lorsque le nombre de motifs séquentiels augmente. La taille de mémoire utilisée est également calculées dans les trois directions. Pour cela, nous créons une matrice de similarité de dimension $(n \times n)$ où n représente le nombre de motifs séquentiels. Notre mesure de similarité n'étant pas symétrique, nous calculons la totalité de la matrice au lieu d'un calcul diagonal. Le temps du calcul de la matrice de similarité est le temps nécessaire pour faire $n \times n$ comparaisons de

similarité avec notre mesure où n est le nombre de motifs séquentiels dans le jeu de données. Nous étudions aussi la taille de la mémoire utilisée lors du calcul de matrice de similarité. Nos résultats montrent que notre mesure de similarité se calcule très rapidement même quand il y a des boucles de conflits. Elle pourra donc être utilisée au sein d'autres algorithmes comme le clustering ou l'extraction de motifs séquentiels, sans influencer le temps d'exécution.

L'expérimentations sont effectuées sur une machine ayant un CPU Intem 2GHz avec 2Go de mémoire vive sous le système d'exploitation Linux Ubuntu. La mesure de similarité est développée sous Java 5.

Jeu de données Nous réalisons les expérimentations sur deux types de jeu de données différents :

- Motifs séquentiels fréquents,
- Séquences de données.

Les motifs séquentiels fréquents sont extraits à partir de données synthétiques. La base de données synthétique est générée par le générateur de données IBM quest[1]. Les motifs séquentiels sont extraits par méthode PrefixSpan. Ensuite, nous avons choisi les ensembles de motifs séquentiels selon l'objectif de l'expérimentation (le nombre des itemsets, le nombre d'items et le nombre de motifs séquentiels dans le jeu de données). Dans la deuxième étape de notre expérimentation, nous avons décidé de faire une analyse sur les séquences de données(cf. sous section 1.1 de la section 1.2). Dans les séquences de données, il existe des items et itemsets (transactions) non fréquents. Sur les séquences de données nous avons alors plus de chance pour avoir des conflits lors de mapping. L'utilisation des séquences de données nous permet de vérifier l'influence des conflits sur le temps d'exécution.

Sur les tests de l'influence du nombre d'itemsets et du nombre d'items sur le temps d'exécution et la taille de mémoire utilisée, les jeux de données sont constitués de 1000 motifs séquentiels (séquences de données). À chaque étape, nous calculons la matrice de similarité, c-à-d. nous réalisons 1000000 comparaisons de similarité (matrice(1000×1000)). Le temps d'exécution et la taille de mémoire donnés sur les courbes présentent le temps d'exécution et la taille de mémoire utilisée pour calculer 1000000 comparaisons.

[1] $www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html$

6.2 Résultats

La figure 6.1 représente l'évolution du temps d'exécution de 1000000 comparaisons par rapport au nombre d'itemsets par séquence. Sur figure 6.1 la courbe rouge montre la taille de la mémoire utilisée. Selon les courbes, la taille de mémoire utilisée ne change pas considérablement. Le temps de calcul de la matrice de similarité quand il y a 10 itemsets par séquence est satisfaisant (116 sec). Cela veut dire que le temps de calcul de similarité de deux motifs séquentiels ayant chacun 10 itemsets est égal à 116 μSec.

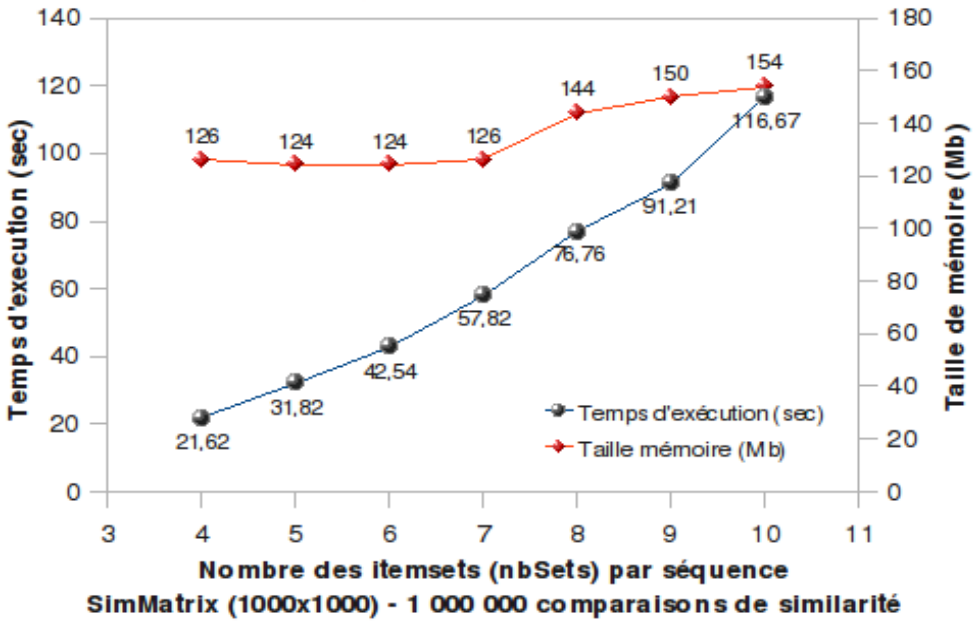

FIG. 6.1 – Temps de calcul et taille de mémoire en fonction du nombre d'itemsets

Notre expérimentation montre que le nombre d'items par séquence n'influence pas le temps de comparaison de similarité autant que le nombre d'itemsets par séquence. Sur la figure 6.2 le temps de comparaison de 1000000 similarités lorsqu'il y a 20 items par séquence est très petit (54 sec). Donc le temps nécessaire pour calculer la similarité de deux motifs séquentiels contenant 20 items chacun est égal à 54 μSec.

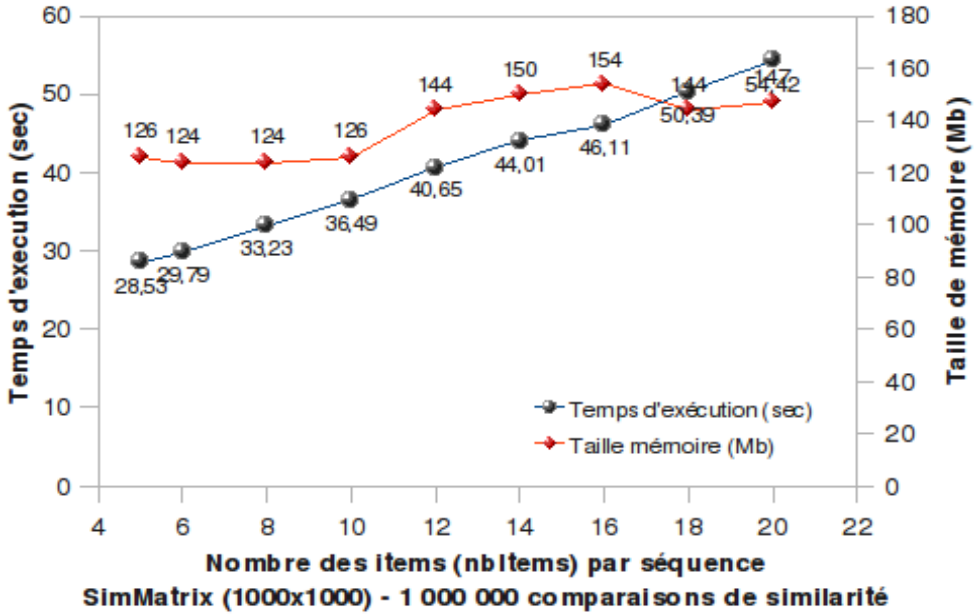

FIG. 6.2 – Temps de calcul et taille de mémoire en fonction du nombre d'items par séquence

Sur la figure 6.3 nous montrons le temps de calcul de matrice de similarité, en augmentant le nombre de séquences. Dans chaque cas, il y a $n \times n$ comparaisons de similarité où n est le nombre de séquences dans le jeu de données. Nous réalisons ce test sur trois types de séquences : les séquences ayant 5 itemsets, ayant 7 itemsets et les séquences ayant 9 itemsets. Dans le cas où il y a 5000 séquences (c-à-d. 25000000 de comparaisons de similarité) et chaque séquence contient 9 itemsets, le temps d'exécution est seulement 1974 sec.

La figure 6.4 représente la taille de la mémoire utilisée sur les mêmes jeux de données (augementation du nombre de séquences). Elle montre aussi que la taille de la mémoire utilisée est plus influencée par l'augmentation du nombre de séquences que par le nombre d'itemsets par séquence.

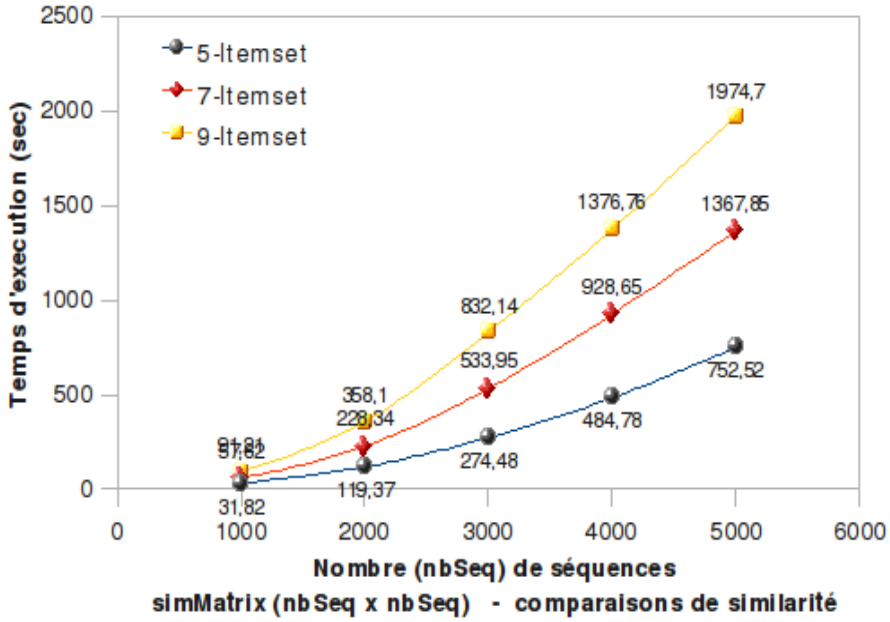

FIG. 6.3 – Temps de calcul de la matrice de similarité en fonction du nombre de séquences

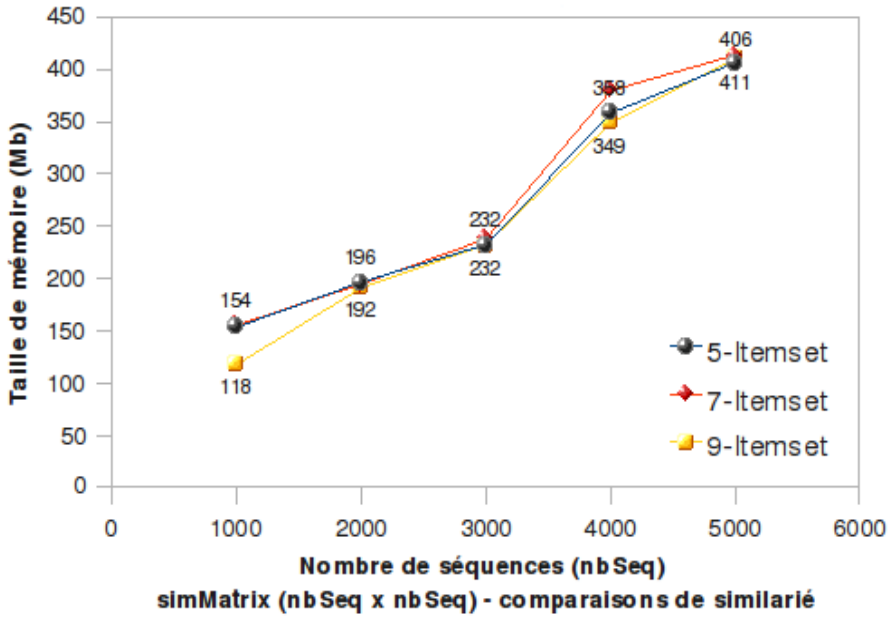

FIG. 6.4 – Taille de mémoire utilisée lors de calcul de la matrice de similarité en fonction du nombre de séquences

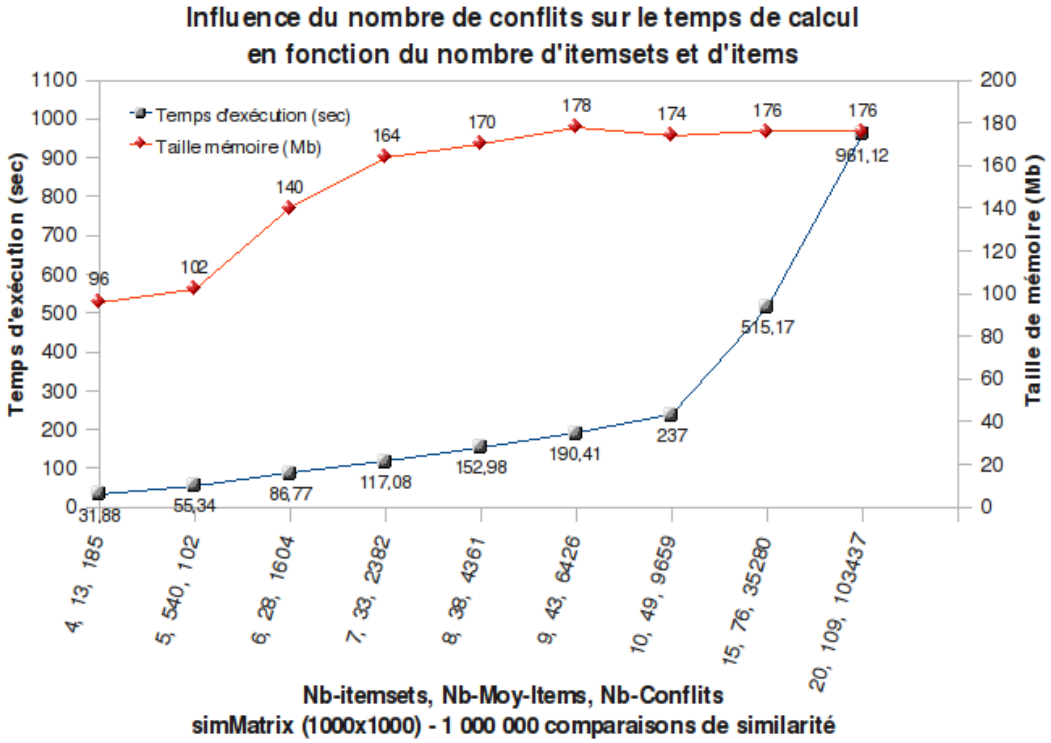

FIG. 6.5 – Influence des conflits sur le temps de calcul de la mesure de similarité

Sur la figure 6.5 nous montrons le résultat de l'expérimentation sur les séquences de données. Pour chaque cas, nous avons marqué le nombre de conflits résolus lors du calcul de la matrice de similarité. L'axe X représente les différents jeu de données. Pour chacun, le nombre d'itemsets et le nombre moyen d'items par séquences sont marqué. Il existe 1000 séquences dans chaque jeu de données (1000000 comparaisons de similarité). Les courbes représentent le temps d'exécution de la matrice de similarité pour chaque cas et la taille de mémoire utilisé. Dans le cas où il y a 20 itemsets et en moyenne 109 items par séquence et 103437 conflits résolus, le temps d'exécution de 1000000 comparaisons de similarité est égal à 961 sec.
Nous remarquons que la taille de la mémoire utilisée est quasiment constante.

6.3 Discussion

Les expérimentations justifient que notre mesure de similarité est utilisable au sein d'autres algorithmes sans les ralentir. En effet son calcul s'exécute assez rapidement même quand il y a beaucoup de conflits lors du mapping. La taille de la mémoire utilisée est influencée seulement par le nombre de séquences dans le jeu de données. Dans un grand jeu de données (5000

séquences ayant 20 itemsets chacun) la taille de la mémoire utilisée est égale à 176 Mo. Nous avons défini une mesure de similarité pour les motifs séquentiels prenant en compte les caractéristiques et la sémantique des motifs séquentiels qui se calcule rapidement et qui ne prend pas beaucoup de mémoire.

Conclusions et Perspectives

Conclusions

Dans ce projet, nous avons défini une approche de détection d'anomalies. Notre approche utilise des motifs séquentiels fréquents pour modéliser les comportements généraux sur un réseau. Nous avons réalisé un clustering de motifs séquentiels pour regrouper les comportements similaires. Cela nous permet de créer des profils de comportements (1ère phase). Dans la 2ème phase de notre approche (évaluation), nous cherchons à identifier les intrusions par les déviation par rapport aux comportements généraux. Ces déviations sont identifiées en comparant les nouveaux motifs séquentiels extraits dans la phase d'évaluation avec les motifs séquentiels représentant des profils de comportements généraux.

Pour le clustering des motifs séquentiels et la comparaison des motifs séquentiels dans la phase d'évaluation, nous avons défini une mesure de similarité adaptée aux motifs séquentiels. Notre mesure prend en compte toutes les caractéristiques des motifs séquentiels et notamment la sémantique des motifs séquentiels. La degré de similarité est composé de deux scores. Ces scores mesurent la similarité des motifs séquentiels à la fois au niveau des itemsets et de leurs positions dans les séquences mais aussi au niveau des items contenus dans les itemsets correspondant. Les expérimentations montrent qu'elle se calcule très rapidement même lorsqu'il y a beaucoup de séquences ayant plusieurs itemsets.

Notre mesure de similarité peut être utilisée pour d'autres applications que le clustering de motifs séquentiels. Plusieurs domaines et méthodes sont envisageables, comme l'extraction des motifs séquentiels sous contrainte de similarité, la compression des motifs séquentiels, la visualisation des motifs similaires etc. Cette mesure est une mesure asymétrique qui permet de faire les comparaisons directionnelles (adaptée aux cas où on compare des motifs séquentiels avec un motif de référence comme l'extraction de motifs séquentiels sous contrainte de similarité). En cas de besoin d'une mesure symétrique, elle peut être définie en considérant la moyenne des degrés de similarité pour les deux directions

Perspectives

Ce travail porte d'une part sur la détection d'anomalies et d'autre part sur la définition d'une mesure de similarité pour les motifs séquentiels.

À court terme :

Nous allons expérimenter notre approche de la détection d'anomalies sur des données réelles. Pour cela, il faut implémenter des méthodes d'extraction d'attributs à partir des logs et Un pré-traitement sur les attributs pour les préparer afin de fouiller. Concernant la mesure de similarité, nous envisageons de considérer une version symétrique de notre mesure. Cela nécessite une modification au niveau des mappings des itemsets. Nous allons ensuite comparer les résultats de la mesure symétrique avec celui de la mesure asymétrique.

À plus long terme :

Notre approche de détection d'anomalies utilise les motifs séquentiels pour modéliser les comportements. Pour l'extraction de motifs séquentiels, il nous faut d'identifier les notions "client" et "transaction" sur les logs d'activités. Chaque définition de ces notions permet de modéliser un type de comportements. Nous allons évaluer les différentes manières de définir de ces notions en fonction du format de logs.

Nous envisageons aussi adapter d'autres méthodes de clustering notamment le clustering dynamique aux motifs séquentiels. Le clustering peut ne pas être précis à cause du bruit dans le jeu de données. C'est pourquoi nous implémenterons des méthodes de post-traitement pour améliorer les résultats de clustering.

Nous allons trouver d'autres applications dans lesquelles notre mesure de similarité peut être utilisée. Premièrement, dans le domaine de fouille de données biologiques, les motifs séquentiels extraits sont volumineux. D'ailleurs, pour l'expert, tous les motifs séquentiels extraits à partir des données biologiques ne sont pas intéressants. La solution envisageable est de fournir une visualisation des motifs séquentiels pour pouvoir observer les motifs séquentiels souhaités. Pour cela, l'expert choisit un motif séquentiel comme référence et l'outil de visualisation lui présente d'autres motifs séquentiels ressemblant parmi tous les motifs extraits. Notre objectif est d'intégrer cette mesure de similarité au sein de ce projet de visualisation des motifs séquentiels.

Bibliographie

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 26–28 1993.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[BCH+01] Eric Bloedorn, Alan D. Christiansen, Willian Hill, Clement Skorupka, Lisa M. Talbot, and Jonathan Tivel. Data mining for network intrusion detection : How to get started, August 2001.

[BGFI+98] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, Eugene H. Spafford, and Diego Zamboni. An architecture for intrusion detection using autonomous agents. In *ACSAC*, pages 13–24, 1998.

[Bru04] S Terry Brugger. Data mining methods for network intrusion detection. Technical report, University of Califorinia, Davis, 2004.

[CA03] M. V. Mahoney Chan, P. K. and M. H. Arshad. *Managing Cyber Threats : Issues, Approaches and Challenges, Chapter 1 : Learning Rules and Clusters for Anomaly Detection in Network Traffic*. 2003.

[Chi01] A. Chittur. *Model generation for an intrusion detection system using genetic algorithms*. PhD thesis, Ossining High School. In cooperation with Columbia Univ., 2001.

[CMB02] Matthieu Capelle, Cyrille Masson, and Jean-François Boulicaut. Mining frequent sequential patterns under a similarity constraint. In *IDEAL*, pages 1–6, 2002.

[DD00] J. E. Dickerson and J. A. Dickerson. Fuzzy network profiling for intrusion detection. In *In Proc. of NAFIPS 19th International Conference of the North American Fuzzy Information Processing Society, Atlanta*, page 301–306. North American Fuzzy Information Processing Society (NAFIPS), July 2000.

[DEK+02] Paul Dokas, Levent Ertoz, Vipin Kumar, Aleksandar Lazarevic, Jaideep Srivastava, and Pang-Ning Tan. Data mining for network intrusion detection. University of Minnesota, Minneapolis, MN 55455, USA, 2002.

[Den87] Dorothy E. Denning. An intrusion-detection model. *IEEE Trans. Software Eng.*, 13(2) :222–232, 1987.

[DGR] Luca Didaci, Giorgio Giacinto, and Fabio Roli. Ensemble learning for intrusion detection in computer networks.

[Fio07] Céline Fiot. *Extraction de séquences fréquentes : des données numériques aux valeurs manquantes*. PhD thesis, Université Montpellier II, septembre 2007.

[GK01] Valerie Guralnik and George Karypis. A scalable algorithm for clustering sequential data. In *ICDM '01 : Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 179–186, Washington, DC, USA, 2001. IEEE Computer Society.

[Har75] J. Hartigans. *clustering algorithms*. John Wiley and Sons, Inc., 1975.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining : Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[Kum95] S. Kumar. *Classification and Detection of Computer Intrusions.* PhD thesis, Purdue Univ., West Lafayette, IN, 1995.

[Kum04] Hye-Chung(Monica) Kum. *Approximate Mining of Consensus Sequential Patterns*. PhD thesis, University of North Carolina, August 2004.

[L.97] Carbone P. L. *Data mining or knowledge discovery in databases : An overview. In Data Management Handbook.* New York : Auerbach Publications, 1997.

[Lan00] T. Lane. *Machine Learning Techniques for the Computer Security Domain of Anomaly Detection*. PhD thesis, Purdue University, August 2000.

[LS98] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[LS00] Wenke Lee and Salvatore J. Stolfo. A framework for constructing features and models for intrusion detection systems. *Information and System Security*, 3(4) :227–261, 2000.

[LSM] Wenke Lee, Salvatore J. Stolfo, and K. Mok. Algorithms for mining system audit data. In T. Y. Lin and N. Cercone, editors, *Data Retrieval and Data Mining*. Kluwer Academic Publishers.

[LSM98] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Mining audit data to build intrusion detection models. In *Knowledge Discovery and Data Mining*, pages 66–72, 1998.

[LSM99] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[LSM00] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Adaptive intrusion detection, a data mining approach. *Artificial Intelligence Review*, 14(6) :533–567, 2000.

[Lue99] J. Lue. Integrating fuzzy logic with data mining methods for intrusion detection. Master's thesis, Mississippi State Univ, 1999.

[Lun90] Teresa F. Lunt. Detecting Intruders in Computer Systems. In *Proceedings of the Sixth Annual Symposium and Technical Displays on Physical and Electronic Security*, 1990.

[LX01] Wenke Lee and Dong Xiang. Information-theoretic measures for anomaly detection. In *Proc. of the 2001 IEEE Symposium on Security and Privacy*, pages 130–143, may 2001.

[MHL94] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Netword intrusion detection. *IEEE Network*, 8(3) :26–41, 1994.

[Moe00] Pirjo Moen. *Attributsn, Event Sequence, and Event Type Similarity Notions for Data Mining*. PhD thesis, University of Helsinki, Finland, 2000.

[Ner00] Filippo Neri. Comparing local search with respect to genetic evolution to detect intrusion in computer networks. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 238–243, La Jolla Marriott Hotel La Jolla, California, USA, 6-9 2000. IEEE Press.

[PES01] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering, 2001.

[PLT07] Marc Plantevit, Anne Laurent, and Maguelonne Teisseire. Extraction d'outliers dans des cubes de données : une aide à la navigation. In *3èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2007), Poitiers*, volume B-3 of *RNTI*. Cépaduès, Address = Toulouse, Pages = 113-130, Juin 2007.

[SHM02] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2) :105–136, 2002.

[SM07] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). Technical Report SP800-94, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, U.S. Department of Commerce, February 2007.

[SPM99] Chris Sinclair, Lyn Pierce, and Sara Matzner. An application of machine learning to network intrusion detection. In *ACSAC '99 : Proceedings of the 15th Annual Computer Security Applications Conference*, page 371, Washington, DC, USA, 1999. IEEE Computer Society.

[SSM06] Abhinav Srivastava, Shamik Sural, and Arun K. Majumdar. Weighted intra-transactional rule mining for database intrusion detection. In *PAKDD*, pages 611–620, 2006.

[SZ02] Karlton Sequeira and Mohammed Javeed Zaki. Admit : anomaly-based data mining for intrusions. In *KDD*, pages 386–395, 2002.

[Tan05] Doru Tanasa. *Web Usage Mining : Contributions to Intersites Logs Preprocessing and Sequential Pattern Extraction with Low Support*. PhD thesis, Université de Nice Sophia Antipolis, Juin 2005.

[Zad65] L.A. Zadeh. Fuzzy sets. information and control. pages 8 (3) 338–353, 1965.