- Université de Versailles St-Quentin en Yvelines -

- Thèse -

Présentée et soutenue publiquement le 07 Janvier 2002, par :

Florent Masseglia

pour obtenir le grade de

Docteur de L'Université

Discipline : Informatique
École doctorale : UVSQ

Algorithmes et applications pour l'extraction de motifs séquentiels dans le domaine de la fouille de données : de l'incrémental au temps réel

JURY

M. Georges Gardarin	**Directeur de thèse**
M. Stefano Ceri	**Rapporteur**
M. Patrick Valduriez	**Rapporteur**
Mme Maguelonne Teisseire	
Mme Danièle Hérin	
M. Pascal Poncelet	

"L'expérience est une lanterne attachée dans notre dos... elle n'éclaire que le chemin parcouru."

Confucius

Remerciements

Cette thèse s'est déroulée au sein du laboratoire d'informatique de robotique et de micro-électronique de Montpellier, en convention avec le laboratoire PRiSM de Versailles St-Quentin. Ce travail a été financé par une allocation de recherche du ministère de l'éducation nationale, de la recherche et de la technologie du gouvernement Français.

Je tiens à exprimer mes plus vifs remerciements à Georges Gardarin pour avoir accepté d'être mon directeur de thèse. Les discussions que nous avons pu avoir ont su, malgré la distance, être fréquentes, enrichissantes et profitables pour l'orientation et la rigueur de ce travail.

Messieurs Stefano Ceri et Patrick Valduriez m'ont fait l'honneur d'être les rapporteurs de ce mémoire. Je leur en suis particulièrement reconnaissant et les remercie chaleureusement pour le temps et l'attention qu'ils ont consacrés à ce travail.

Je remercie Danièle Hérin, pour avoir accepté de faire partie de mon jury et pour avoir suivi de façon attentive et engagée ma progression depuis mon stage de DEA. Cette première expérience s'était déjà déroulée au sein de son équipe dans le département ARC, avec un accueil et une sympathie qui se sont confirmés lors de mes trois années de thèse.

Je tiens à remercier Maguelonne Teisseire pour avoir accepté le co-encadrement de cette thèse. Qu'elle reçoive toute ma gratitude pour m'avoir laissé si souvent le champ libre et m'avoir fait confiance pour mener à bien les travaux dans lesquels je me suis lancé. Pour m'avoir laissé m'entêter tout en m'apportant son expérience et ses conseils judicieux quand j'ai fait des choix parmi les nombreux sujets auxquels l'équipe me confrontait, je lui suis extrêmement reconnaissant.

D'une certaine façon, cette thèse a vu le jour il y a maintenant huit ans, à l'Université d'Aix-Marseille, au sein du département Informatique de l'IUT d'Aix en Provence. Un enseignant particulièrement enthousiaste et confiant, a su m'accorder son attention quand j'exprimais mon désir de poursuivre mes études dans la voie de la recherche et de l'enseignement. Pascal Poncelet, s'est alors montré généreux a ce point qu'il a, depuis cette rencontre à l'IUT, gardé le contact avec moi tout au long de mon cursus universitaire pour m'expliquer les rouages, les tenants et les aboutissants d'un parcours menant jusqu'au doctorat. Après qu'il ait assuré l'encadrement de mon travail de DEA, j'ai su que ma thèse, sous son co-encadrement, ne pouvait que bénéficier d'un enthousiasme, d'un sérieux, d'une productivité et d'une qualité scientifique indiscutables. Pour m'avoir épaulé et conseillé pendant tout ce temps, pour avoir donné à mon parcours cette impulsion et ce rythme qui lui ont permis d'avancer aussi fort jusqu'à ce jour et probablement encore pour longtemps, merci Pascal.

Les services administratifs des laboratoires LIRMM et PRiSM m'ont été d'un grand secours lorsque des difficultés extra-scientifiques se sont présentées. Je suis heureux d'avoir pu compter sur le soutien, la sympathie et l'efficacité de leurs membres et je remercie particulièrement Josette Durante, Nadine Thilloy et Chantal Ducoin pour en avoir fait preuve.

Je tiens enfin à remercier ma famille pour son soutien et sa patience, merci à Corinne et merci à mes amis pour avoir accepté mon manque de disponibilité : Bruno, Fred, Jeff, Jeremy, Mu, Yannic et les autres...

Enfin, merci à ceux que j'ai oubliés mais qui sauront ne pas m'en vouloir.

Résumé

L'extraction de connaissances dans les bases de données (ECD) permet d'exploiter le fait que des informations soient stockées en grande quantités, afin d'y trouver des schémas nouveaux et potentiellement utiles. Ce processus comporte trois étapes principales : le prétraitement des données, la fouille de données et l'interprétation des résultats. Nous proposons, dans ce mémoire, une étude qui s'étend du prétraitement jusqu'à la fouille de données et qui considère les applications que l'on peut envisager pour ces techniques. La fouille de données concerne les algorithmes qui ont pour but d'analyser les données et qui sont en contact direct avec elles. La recherche de motifs séquentiels est une des problématiques de la fouille de données. Elle consiste à trouver des enchaînements d'événements récurrents, parmi un ensemble d'enregistrements d'enchaînements plus longs. Pour améliorer la qualité des séquences extraites, les contraintes de temps ont pour but d'affiner les résultats en précisant quels écarts de temps dans les enchaînements analysés sont trop courts ou trop longs. Les premiers algorithmes que nous proposons, i.e. PSP et GTC, ont pour but de résoudre la problématique des motifs séquentiels ainsi que leur généralisation par les contraintes de temps. La rapidité avec laquelle les données sont stockées, laisse comprendre la faible pérénité des résultats obtenus. Il est donc indispensable de proposer des solutions à cet état de fait, en intégrant des méthodes incrémentales pour l'extraction de motifs séquentiels. L'algorithme ISE, proposé dans ce mémoire, est conçu dans ce sens. Les applications que nous avons implémentées, sont liées de façon plus ou moins intuitive aux motifs séquentiels. Elles permettent d'extraire des sous-phrases communes à un ensemble de textes ou encore de découvrir des structres fréquentes dans des données semi-structurées. Un domaine particulier a gagné notre attention : le Web Usage Mining. Les techniques de ce domaine visent à analyser le comportement des utilisateurs d'un site web. Cette analyse se base sur les enregistrement du fichier access log. Notre méthode est capable d'analyser ces fichiers, mais aussi de prendre en compte les résultats lors de connexions nouvelles des utilisateurs, afin d'adapter le contenu du site en fonction du comportement de chacun. Cependant, les possibilités offertes par le contexte du web nous semblent dépasser le cadre de l'exploitation qui en est faite à l'heure actuelle. Nous proposons donc, au travers des notions de Web Usage Mining Inter-Sites et d'analyse de comportement en temps réel, une augmentation de la connaissance acquise par un processus de Web Usage Mining. Pour terminer notre étude, nous proposons de revenir sur l'étape de prétraitement des données, dans le processus d'ECD, afin de mettre à profit l'expérience apportée par les applications que nous avons traitées. Notre objectif est alors de mettre au point un prétraitement automatique ou semi-automatique des données et d'intégrer le résultat dans *L*eitmotiV, la plateforme d'ECD que nous avons développée. L'aspect délicat de cette analyse provient du caractère obligatoire de la qualité de ce prétraitement.

Mots clefs : ECD, fouille de données, motifs séquentiels, contraintes de temps, fouille de données incrémentale, text mining, schéma mining, web usage mining, inter-sites, temps réel, motifs longs, fouille de données intéractive, prétraitement des données.

Avant-propos

Le stade des balbutiements, que les techniques liées à la fouille de données ont vécu au début des années 90, connaît une évolution telle, qu'il s'ouvre désormais vers la maturité et la stabilité de ce domaine. Si ce dernier semble promettre un avenir fourni en problématiques, ou en interrogations, il s'est construit des bases solides qui nous permettent d'envisager la fouille de données avec plus de recul, et donc probablement plus d'efficacité.

Les chiffres, auxquels nous sommes confrontés régulièrement dans ce domaine, varient peu et restent aussi impressionants qu'ils l'étaient quand la progression des moyens de stockage nous a conduit vers la mise au point de méthodes pour exploiter, analyser, comprendre et peut-être prévoir les données qu'ils engrangeaient. Des informations telles que la multiplication par deux, tous les cent jours, du trafic sur internet (selon [INS]), ou encore telles que le nombre de téléphones mobiles en UE, qui est passé de 21 millions en 1995 à 146 millions en 1999 (toujours selon [INS]), nous rappellent bien sûr que l'efficacité des méthodes de fouille de données à mettre en place reste une priorité pour la recherche.

Cependant, ces chiffres, s'ils sont toujours la cause d'efforts destinés à les maîtriser, sont à notre sens le reflet du besoin de prendre en compte le contexte dans lequel ces efforts sont déployés. Une des informations que nous pouvons considérer (selon [SBKFF]) dit que 'Yahoo' a 166 millions de visiteurs par jour générant 48GB de flot de données de type "clickstream" [1] par heure. Cette quantité de données nous apparaît bien sûr comme un défi du point de vue de la fouille de données. Il suffit, en effet, d'imaginer les moyens à mettre en œuvre pour, par exemple, analyser les navigations qui ont été pratiquées sur le site dans une période finie. Mais si l'on envisage le fait que cette quantité de données (de "clickstreams") abrite des utilisateurs, des ordinateurs reliés au site, alors cette considération modifie la perception du problème en nous faisant prendre en compte un facteur de première importance : le contexte.

Soucieux de garder à l'esprit cette notion de contexte, nous voulons prendre en considération différents facteurs, liés aux nouvelles technologies de stockage des données, à la nouvelle économie, ou encore aux techniques de fouille de données elles-mêmes.

1. "clickstream": enregistrement de l'activité de l'utilisateur sur le site (par exemple l'URL demandée par un visiteur grâce à un lien hypertexte).

Pour analyser une base de données volumineuse, comme celle qui recense les achats dans un supermarché par exemple, les méthodes les plus efficaces sont mises au point et encore améliorées ; mais le cadre du supermarché, comme celui du commerce en général, implique un besoin marketing typique : la fidélisation. Comment permettre de s'adapter aux clients, si l'on ne peut pas analyser leur comportements de façon durable dans le temps ? C'est une notion qui conduit à la prise en compte du facteur *temps*[2] dans l'analyse des données et que nous allons traiter tout au long de ce mémoire. En effet l'exemple utilisé ici, permet de mettre en avant l'importance du facteur temps (ou l'organisation des événements), mais ce facteur a un caractère indispensable dans bien d'autres domaines d'application.

Ce même supermarché cumule jour après jour des quantités de données, aussi grandes que les chiffres présentés plus haut le laissent imaginer. Cet élément du contexte, nous conduit à penser que l'analyse à fournir pourrait gagner à se doter d'une capacité *incrémentale* dans la fouille à entreprendre.

Le succès du commerce électronique[3] sera traité avec une attention particulière, car il conforte le besoin d'utiliser les techniques issues du domaine de la fouille de données. Le résultat apparaît alors comme un domaine à part entière : *l'analyse du comportement des utilisateurs d'un site web*. Cet engouement pour la nouvelle génération du commerce est également à l'origine d'une prolifération de sites, qui voient grandir les quantités d'informations qu'ils diffusent, sans pour autant les maîtriser. Parmi les éléments qui peuvent conduire à cette maîtrise, la découverte d'une *structure* servant de base à leur réorganisation peut être capitale.

Enfin, l'étendue grandissante des types de données traitées par les méthodes de fouille est telle que le responsable d'un processus de fouille de données peut se trouver confronté à des sources systématiquement différentes. Le *prétraitement* de ces données (étape qui est considérée comme un pilier du processus d'extraction de connaissances) mérite une étude attentive. Si cette étude peut se solder par une plus grande facilité d'adaptation en fonction de la source, alors l'administrateur de la fouille pourra relâcher ses efforts sur ce point, au profit de problèmes plus difficiles à automatiser.

L'une des orientations adoptées par ce mémoire consiste à reconsidérer à la hausse l'intérêt que l'on peut porter à ces informations, pour ajouter aux notions de technologies efficaces celle d'une analyse mieux adaptée aux domaines considérés. Nous proposons d'aborder, de définir et de résoudre quelques problématiques dans le domaine de la fouille de données, en considérant le contexte dans lequel les techniques de fouille sont amenées à être produites, mais aussi à évoluer. Nous avons pour but de rester vigilants vis à vis du contexte dans lequel s'inscrivent les méthodes proposées, tout en focalisant nos efforts sur leur indispensable efficacité.

2. La temporalité joue un double rôle dans ce mémoire. Le premier aspect est de distinguer la façon dont les événements s'enchaînent. Le second demande de répondre à des besoins de contraintes sur les écarts de temps dans ces enchaînements.

3. Selon [IPS] : "La majorité des européens interrogés juge aujourd'hui qu'internet est "très bien" ou "assez bien adapté" pour l'achat d'un billet d'avion ou de train (61%), pour l'achat de livres, disques ou cassettes (58%), de vacances (49%) ou de titres en bourse (44%)".

Table des matières

Chapitre 1

Introduction

Alors que nous progressons, à grande vitesse, dans l'ère de l'information numérique, une contradiction est née, reflet de la distance séparant la surabondance des données et notre aptitude à les rendre significatives.
Il est aujourd'hui facile de constater que les bénéfices de l'enregistrement à grande échelle ont motivé les efforts technologiques, et donc notre capacité à stocker, exploiter, véhiculer et échanger les informations de manière massive. Du groupe financier au centre de recherches médicales, en passant par la compagnie de télécoms, l'information circule et s'empile à un rythme qui ne faiblit jamais. Les moyens créant les besoins, cette nouvelle caractéristique des bases de données offre aujourd'hui des possibilités que les outils traditionnels, par la nature de leurs fonctionnalités, ne permettent pas d'exploiter [Com96]. Il est apparu, en effet, que pour profiter de cette richesse, l'intérêt n'est plus exclusivement de trouver les propriétés de l'information que l'on sait, par avance, intéressante mais, à l'inverse, de trouver l'information intéressante à partir des propriétés que l'on estime révélatrices. Pour répondre à cet intérêt et proposer des solutions issues des travaux sur les bases de données, l'intelligence artificielle et les statistiques, les techniques d'ECD[1] et de fouille de données[2], ont pour but la recherche et l'interprétation d'informations importantes, dans des bases de données de plus en plus volumineuses.

1.1 L'extraction de connaissances dans les bases de données

Le processus d'ECD désigne l'ensemble des opérations qui permettent d'exploiter le fait que les données puissent être stockées de manière massive avec facilité et rapidité. Dans la définition d'origine, **l'ECD est un processus non trivial qui consiste à identifier, dans les données, des schémas nouveaux, valides, potentiellement utiles et surtout compréhensibles et utilisables** [FPSS96]. Cette définition peut garder un aspect légèrement flou, principalement à cause de l'utilisation du terme "potentiellement". Il faut être conscient du fait que le processus d'ECD a pour but d'exploiter le fait que les données sont stockées en grande quantité, plutôt que de s'intéresser aux données elles-mêmes. Que ces données soient issues des codes barres inscrits sur les produits vendus par un supermarché, ou bien d'un site web qui recense les activités de ses utilisateurs, leur quantité abondante est leur dénominateur commun. C'est cette quantité que l'ECD veut exploiter. Cette exploitation se fera, bien sûr, via les données elles-mêmes, mais l'idée consiste à fournir une réponse à la question que les propriétaires de ces données peuvent se poser : "Un schéma pourrait-il être trouvé, caractérisant les données que j'ai stockées ? ". Cette question peut alors accepter une réponse simple (oui ou non) mais, dans l'affirmative, le "golden nugget" (le "schéma" de la définition d'origine) tant recherché peut connaître plusieurs formes :

Les Règles d'Association

Le problème de la recherche de règles d'association a fait l'objet de nombreux travaux de recherche ces dernières années [AIS93, AS94, BMUT97, HS95, MTV94, Mue95, PBTL99, SON95, Toi96]. Introduit

1. Le sigle ECD (pour *Extraction de Connaissances dans les bases de Données*) est une traduction de l'anglais KDD (pour *Knowledge Discovery in Databases*) qui fut introduit par Piatetsky-Shapiro en 1989 lors d'un workshop de la conférence IJCAI'89.

2. Le terme fouille de données est une traduction de l'anglais *data mining*.

dans [AIS93] à partir du problème du panier de la ménagère ("Market Basket Problem"), il peut être résumé ainsi : étant donné une base de données de transactions (les paniers), chacune composée d'items (les produits achetés), la découverte de règles d'associations consiste à chercher des ensembles d'items, fréquemment liés dans une même transaction, ainsi que des règles les combinant. Un exemple d'association pourrait révéler que "75% des gens qui achètent de la bière, achètent également des couches". Ce type de règles concerne un grand champ d'applications, telles que la conception de catalogues, la promotion de ventes, le suivi d'une clientèle, etc.

Les Motifs Séquentiels

Introduit dans [AS95], les motifs séquentiels peuvent être vus comme une extension de la notion de règle d'association, intégrant diverses contraintes temporelles. La recherche de tels motifs consiste ainsi à extraire des ensembles d'items, couramment associés sur une période de temps bien spécifiée. En fait, cette recherche met en évidence des associations inter-transactions, contrairement à celle des règles d'association qui extrait des combinaisons intra-transaction. Dans ce contexte, et contrairement aux règles d'association, l'identification des individus ou objets est indispensable, afin de pouvoir suivre leur comportement au cours du temps. Par exemple, des motifs séquentiels peuvent montrer que "60% des gens qui achètent une télévision, achètent un magnétoscope dans les deux ans qui suivent". Ce problème, posé à l'origine dans un contexte de marketing, intéresse à présent des domaines aussi variés que les télécommunications (détection de fraudes), la finance, ou encore la médecine (identification des symptômes précédant les maladies).

Les Dépendances Fonctionnelles

L'extraction de dépendances fonctionnelles à partir de données existantes est un problème étudié depuis de nombreuses années, mais qui a été abordé récemment avec une "vision" fouille de données. Les résultats obtenus par ces dernières approches sont très efficaces [KMRS92, HKPT98, LPL00, NC00]. La découverte de dépendances fonctionnelles est un outil d'aide à la décision à la fois pour l'administrateur de la base, les développeurs d'application, mais également les concepteurs et intégrateurs de systèmes d'information. En effet, les applications relèvent de l'administration et du contrôle des bases de données, de l'optimisation de requêtes ou encore de la rétro-conception de systèmes d'information.

La Classification

La classification, appelée également induction supervisée, consiste à analyser de nouvelles données et à les affecter, en fonction de leurs caractéristiques ou attributs, à telle ou telle classe prédéfinie [BFOS84, WK91]. Bien connus en apprentissage, les problèmes de classification intéressent également la communauté Base de Données qui s'appuie sur l'intuition suivante : "plus importants sont les volumes de données traités, meilleure devrait être la précision du modèle de classification" [SAM96]. Les techniques de classification sont par exemple utilisées lors d'opérations de "mailing" pour cibler la bonne population et éviter ainsi un nombre trop important de non-réponses. De la même manière, cette démarche peut permettre de déterminer, pour une banque, si un prêt peut être accordé, en fonction

de la classe d'appartenance d'un client.

La Segmentation

La technique de segmentation ressemble à celle de la classification, mais diffère dans le sens où il n'existe pas de classes prédéfinies [JD88] : l'objectif est de grouper des enregistrements qui semblent similaires dans une même classe. De nombreux algorithmes efficaces ont été proposés pour optimiser les performances et la qualité des classes obtenues dans de grandes bases de données. Parmi ceux-ci nous pouvons citer [EKSX96], [GRS98a, GRS98b] et [WG99]. Les applications concernées incluent notamment la segmentation de marché ou encore la segmentation démographique par exemple en identifiant des caractéristiques communes entre populations.

Les Séries Chronologiques

L'objectif est de trouver des portions de données (ou séquences) similaires à une portion de données précise, ou encore de trouver des groupes de portions similaires issues de différentes applications [AFS93, ALSS95, CFaM94]. Cette technique permet par exemple d'identifier des sociétés qui présentent des séquences similaires d'expansion, ou encore de découvrir des stocks qui fluctuent de la même manière.

Les différents types de résultats que nous venons de présenter, sont issus d'une étape du processus global d'ECD : la fouille de données. Les algorithmes employés peuvent alors fournir des règles d'association, des motifs séquentiels, des classes ou segments, etc. Ce résultat dépend de la forme sous laquelle se présentent les données à traiter, mais également de la demande faite par l'utilisateur. En fonction de cette demande, et donc du type de fouille à entreprendre, les données d'origine doivent subir différentes transformations avant d'être disponibles pour une analyse. La fouille de données, quand à elle, demande souvent plusieurs réglages, et donc plusieurs itérations, avant de permettre une découverte intéressante. Enfin, pour déterminer le niveau d'intérêt de cette découverte, une interprétation des résultats obtenus doit être faite. Une fois cette interprétation terminée, l'utilisateur peut enfin savoir si dans la quantité de données qu'il a stockées se cachent des schémas intéressants. Le processus d'ECD est donc un enchaînement d'opérations, toutes aussi importantes les unes que les autres, qui connaît un caractère fortement interactif et itératif. La façon dont ces opérations s'organisent peut être observée à la figure 1.1.

1.2 L'extraction de motifs séquentiels

Comme nous venons de le voir, la problématique de l'extraction des motifs séquentiels dans une base de données, est une sorte d'extension de celle des règles d'association. En effet, la prise en compte de la temporalité dans les enregistrements à étudier permet une plus grande précision dans les résultats, mais implique aussi une plus grande difficulté d'implémentation. En effet les algorithmes mis en œuvre pour résoudre ces problèmes ([AS94], [SA96b], [MTV94], ...), sont basés sur la recherche de sous-mots communs à plusieurs phrases, problème reconnu NP-Complet ([GJ79]). Cependant le spectre des

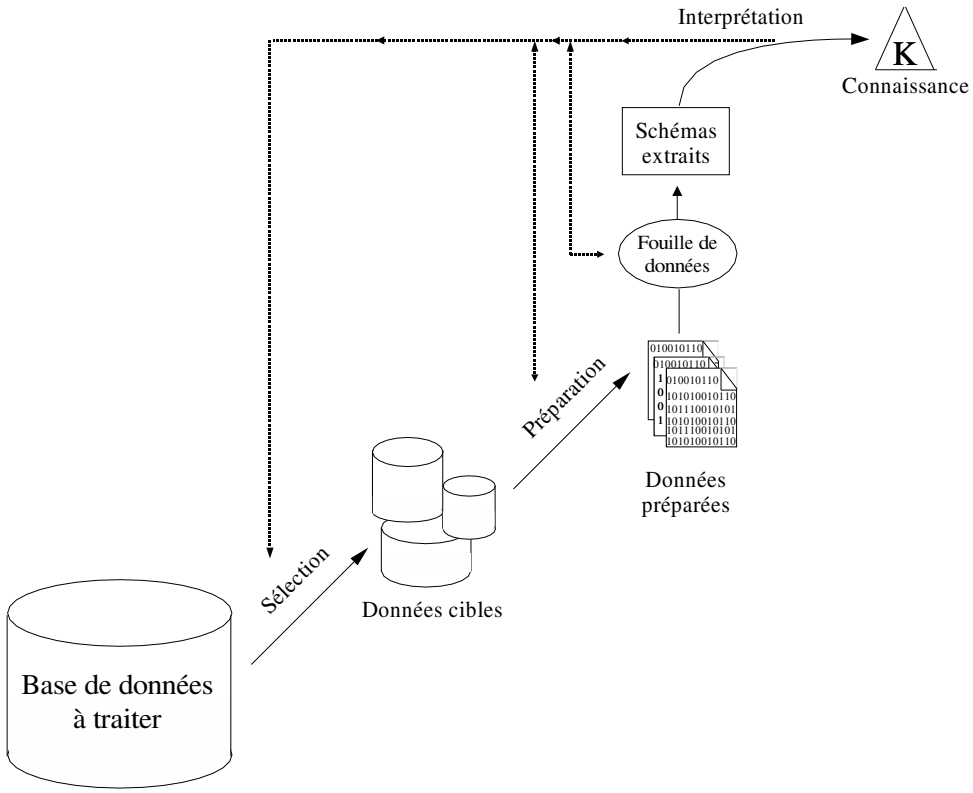

FIG. 1.1 – *Les différentes étapes du processus d'ECD*

domaines d'applications que l'on peut envisager pour les motifs séquentiels est plus large que celui des règles d'association. Cela vient tout d'abord du fait que les motifs séquentiels peuvent être utilisés pour résoudre la problématique des règles d'association. Cela s'explique aussi par la prise en compte de la temporalité, qui permet d'organiser les éléments traités avec une notion d'ordre. Cet ordre, tout d'abord chronologique, peut ensuite être adapté à d'autres domaines en tant que relation d'ordre classique. Nous verrons, par exemple avec la notion de Schema Mining, que l'organisation des éléments n'est pas forcément chronologique mais reste indispensable pour résoudre certaines problématiques.

1.2.1 Présentation et motivations

La recherche de motifs séquentiels se base sur un format de données bien précis, qui relate des événements liés à différents acteurs. L'exemple du supermarché est le plus utilisé pour illustrer ce concept. Cela est, en partie, dû au fait que ce sont des besoins de type marketing (et notamment les supermarchés) qui ont apporté cette problématique. Ce format de données est illustré par la figure 1.2, où l'on peut distinguer les achats de quatre clients, sur une période de quatre jours. Le but d'un algorithme d'extraction de motifs séquentiels sera alors de trouver des comportements fréquents, dans les

achats de ces clients. Le résultat attendu est donc une ou plusieurs *séquences* d'achats, représentant un comportement récurrent. Nous pouvons alors définir le comportement fréquent comme étant un comportement respecté par au moins n clients (n étant considéré comme un minimum[3] de clients qui respectent ce comportement afin que celui-ci soit estimé fréquent). L'exemple 1 illustre la forme sous laquelle un comportement fréquent se présente.

Client	01/10/2001	02/10/2001	03/10/2001	04/10/2001
Client 1	Marteau Bières Clous	Ampoule	Trousse de secours	Scie Manuel du parfait bricoleur Soda
Client 2	CD « little-walter »	Marteau Clous	Trousse de secours Fromage	Scie Manuel du parfait bricoleur
Client 3	Soda Gateaux	Bière Pizzas	Alka-Seltzer	Marteau Clous Scie
Client 4	Marteau Gateaux Clous	Trousse de secours Mouchoirs	Scie Lunettes Manuel du parfait bricoleur	Pizzas Soda Pantoufles

FIG. 1.2 – *Une base de données exemple, limitée à 4 clients*

Exemple 1 *Considérons la figure 1.2, qui regroupe les achats de quatre clients sur une période qui couvre quatre jours. À partir de ces informations, un algorithme de recherche de motifs séquentiels a pour but de trouver des répétitions, dans les enchaînements d'achats décrits par cette base. Avec n (le nombre minimum de clients) fixé à 75% (soit 3 clients sur les 4 de la base). Nous pouvons alors constater qu'un comportement fréquent est : "le client achète dans la même journée* un marteau *et* des clous. *Ensuite (un jour postérieur à cet achat) il achète une* trousse de secours. *Ce comportement se solde ensuite, un jour encore postérieur, par un achat composé d'une* scie *et d'un* manuel du parfait bricoleur.*" L'interprétation de ce comportement étant laissé à l'expert du processus d'ECD, le traitement par l'algorithme d'extraction de motifs séquentiels s'achève sur cette découverte.*

La figure 1.3 propose de visualiser plus clairement le comportement découvert sur la base de données 1.2. Nous pouvons y constater que les références des clients n'y sont plus reportées. Cela illustre le fait que l'extraction de motifs séquentiels ne cherche pas à attribuer un comportement à tel ou tel client, mais bien à extraire un comportement commun à n clients. La notion de date pourrait également disparaître de ce tableau, dans la mesure où l'information à retenir sur la temporalité se limite au changement de jour et non pas aux dates précises.

La méthode Générer-Élaguer

Pour résoudre la question des motifs séquentiels, les premiers algorithmes se sont basés sur la méthode "Générer-Élaguer". Cette méthode, issue des travaux relatifs aux règles d'association, consiste

3. Nous verrons dans la définition formelle que n porte le nom de "support minimum" pour l'algorithme de fouille de données.

01/10/2001	02/10/2001	03/10/2001	04/10/2001
Marteau Bières **Clous**	Ampoule	**Trousse de secours**	**Scie** **Manuel du parfait bricoleur** Soda
CD « little-walter »	**Marteau** **Clous**	**Trousse de secours** Fromage	**Scie** **Manuel du parfait bricoleur**
Soda Gateaux	Bière Pizzas	Alka-Seltzer	Marteau Clous Scie
Marteau Gateaux **Clous**	**Trousse de secours** Mouchoirs	**Scie** Lunettes **Manuel du parfait bricoleur**	Pizzas Soda Pantoufles

FIG. 1.3 – *Mise en relief du motif séquentiel découvert pour la base de la figure 1.2*

à tester des candidats sur la base avant de les déclarer fréquents. Ces candidats sont construits de façon très précise, qui sera détaillée dans l'état de l'art concernant la recherche de motifs séquentiels. Une des caractéristiques de la mise au point des candidats réside dans le côté croissant de la taille des candidats proposés. En effet la méthode part des éléments[4] fréquents, avant de proposer des candidats (séquences) de taille 2. À partir des candidats de taille 2, une passe sur la base permet de compter leur support[5] afin de les déclarer fréquents ou pas. Les candidats devenus fréquents sont alors utilisés pour générer des candidats de taille 3 et le processus recommence jusqu'à ce qu'aucun autre candidat ne puisse être généré, ou qu'aucun fréquent ne soit plus détecté. La figure 1.4 illustre la façon dont fonctionne la méthode générer-élaguer (une séquence de taille n est une n-séquence, donc les n-candidats sont les candidats de taille n et les n-fréquents sont les séquences fréquentes de taille n).

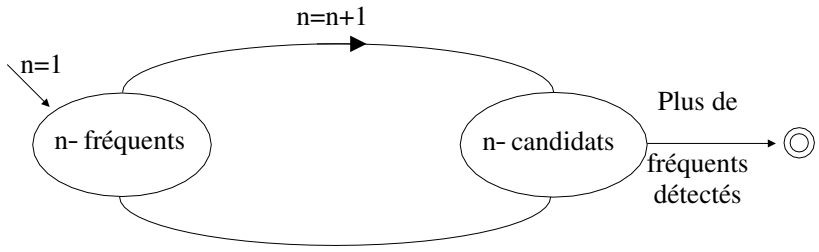

FIG. 1.4 – *Automate implémentant la méthode générer-élaguer*

4. Les éléments sont généralement désignés par le terme "items", que nous utiliserons dans la suite de ce mémoire.

5. Le *support* d'un candidat sera défini plus loin comme le nombre de séquences qui contiennent ce candidat. Dans le cas du supermarché, il s'agit donc du nombre de client qui respectent le comportement décrit par ce candidat.

La structure d'arbre de hachage

Le premier algorithme présenté pour exploiter la méthode générer-élaguer dans le cadre de l'extraction de motifs séquentiels fut donné par [AS95]. Cet algorithme utilise une structure destinée à optimiser la gestion des candidats lors de leur évaluation. En effet le principe d'une passe sur la base consiste à prendre l'enregistrement [6] de chaque client et à le comparer à chacun des candidats. Par exemple avec la base de données de la figure 1.2, si l'on considère le client 2, il faudra comparer la séquences < CD "little-walter" ; Marteau, Clous ; Trousse de secours, Fromage ; Scie, Manuel du parfait bricoleur > avec chacun des candidats proposés par l'algorithme. Pour éviter de comparer cette séquence avec tous les candidats, [AS95] propose de classer ces derniers dans un arbre de hachage. Cet arbre a pour but de filtrer les candidats qui ne peuvent pas être contenus dans la séquence de données étudiée. Une fois ces candidats filtrés, il reste ceux qui peuvent être inclus. Ceux là seulement méritent d'être analysés. L'exemple 2 illustre le fonctionnement de l'arbre de hachage.

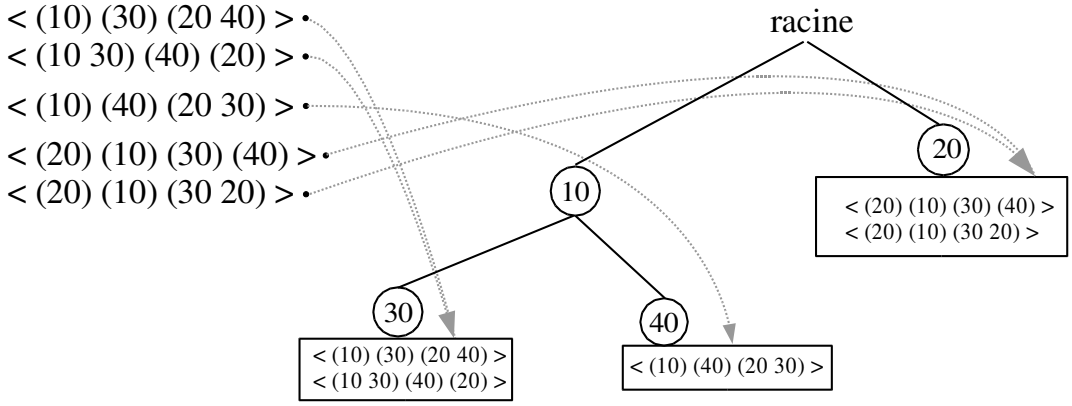

FIG. 1.5 – *Des candidats stockés dans l'arbre de hachage*

Exemple 2 *La figure 1.5 représente un ensemble de candidats et l'arbre dans lequel ils sont stockés. Le candidat < (10) (30) (20 40) > s'interprète "Achat de l'item 10, suivi un jour postérieur par l'achat de l'item 30, suivi un jour postérieur par l'achat simultané des items 20 et 40." L'arbre de hachage dans lequel seront stockés ces candidats est également illustré à la figure 1.5. On peut y constater que l'arbre possède trois feuilles. Ce sont ces feuilles qui stockent les candidats.*

Pour ajouter un candidat dans l'arbre, l'algorithme suit un parcours préfixé de l'arbre en fonction de la séquence candidate. Une fois qu'une feuille est atteinte alors on y ajoute le candidat. Si cette feuille atteint sa taille maximale alors elle est scindée en deux et on redistribue tous les candidats qu'elle contient. Par exemple, la séquence < (10) (30) (20 40) > sera ajoutée selon le principe suivant : le premier item (i.e. 10) de la séquence conduit au sommet 10 (fils de la racine) de l'arbre, le second item du candidat (i.e. 30) conduit sur le fils de 10 dans l'arbre et, comme il s'agit d'une feuille, le candidat

6. Nous définirons plus tard ces enregistrements comme étant les séquences de données (ou *data-sequences*).

y est ajouté.

De la même manière, chaque séquence de données à analyser est projetée dans l'arbre afin de trouver les feuilles susceptibles de contenir un candidat supporté par cette séquence. Par exemple avec l'arbre de la figure 1.5 et la séquence de données < (20) (40) (20 30) > seule la feuille du sommet 20 (fils de la racine) sera atteint par cette séquence. Donc les seuls candidats à être analysés sont ceux contenus par cette feuille et l'algorithme évite de tester ceux contenus dans les deux autres feuilles.

1.2.2 Contribution

Nous estimons que l'arbre de hachage, mis en place par [AS95], présente un défaut qu'il est facile de constater. En effet lors de la recherche des feuilles susceptibles de contenir des candidats inclus dans la séquence analysée, [AS95] ne tient pas compte des changements de jour[7] entre les items de la séquence qui servent à la navigation. Par exemple, avec la séquence < (10 30) (20 40) >, l'algorithme va atteindre la feuille du sommet 30 (fils de 10), alors que cette feuille peut contenir deux types de candidats :

- ceux qui commencent par < (10) (30)... d'un côté
- et ceux qui commencent par < (10 30) ... de l'autre.

Notre but est alors de mettre en place une structure d'arbre préfixé, pour gérer les candidats. L'algorithme PSP ([MCP98, MPC99a]), destiné à exploiter cette structure, sera basé sur la méthode générer-élaguer. Le principe de base de cette structure consiste à factoriser les séquences candidates en fonction de leur préfixe. Cette factorisation, inspirée de celle mise en place par [AS95], pousse un peu plus loin l'exploitation des préfixes communs que présentent les candidats. En effet nous proposons de prendre en compte les changements d'itemsets dans cette factorisation. L'arbre préfixé ainsi proposé ne stocke plus les candidats dans les feuilles, mais permet de retrouver les candidats de la façon suivante : tout chemin de la racine à une feuille représente un candidat et tout candidat est représenté par un chemin de la racine à une feuille. De plus, pour prendre en compte le changement d'itemset, nous avons doté cet arbre de deux types de branches. Le premier type, entre deux items, signifie que les items sont dans le même itemset alors que le second signifie qu'il y a un changement d'itemset entre ces deux items. L'exemple 3 donne une illustration de la structure d'arbre préfixé.

Exemple 3 *La figure 1.6 illustre la façon dont les candidats sont stockés dans l'arbre préfixé, avec les mêmes candidats que ceux présentés à la figure 1.5. La figure propose une comparaison du stockage des candidats selon que l'on utilise la structure de hachage ou d'arbre préfixé. On peut y distinguer les items appartenant à un même itemset (branche en pointillés) ou bien les changements d'itemsets (branche en trait plein).*

La structure préfixée présente alors deux avantages :

- Une **complexité en mémoire** plus faible. On peut constater, en effet, que le nombre d'éléments à utiliser pour représenter l'arbre est inférieur au nombre d'éléments que présentent les candidats au total. Le nombre d'éléments nécessaires pour représenter l'arbre est de l'ordre du nombre

7. Nous verrons dans les définitions qu'une journée d'achat, au sens du supermarché, est aussi appelée *itemset*.

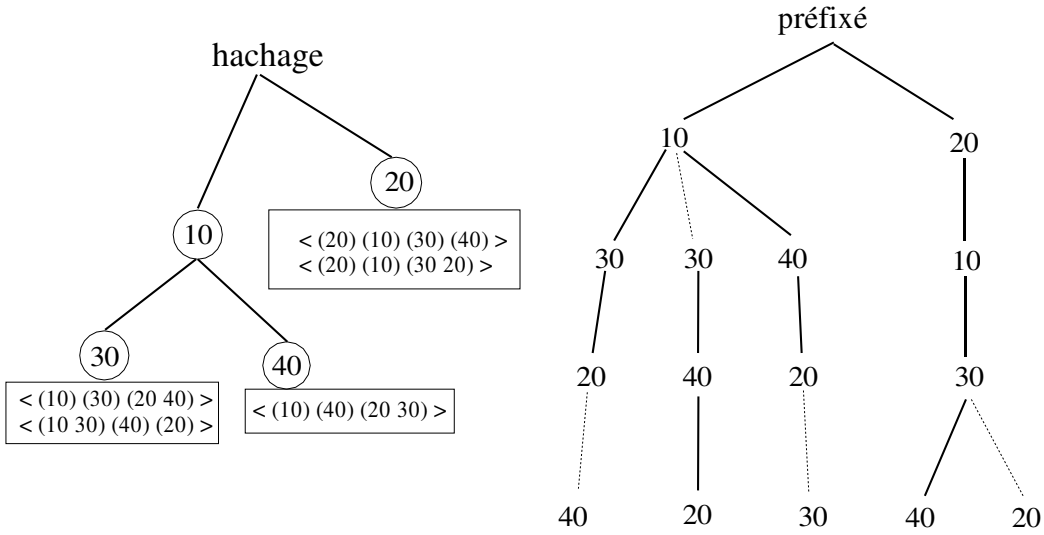

FIG. 1.6 – *Comparaison entre la structure de hachage et l'arbre préfixé*

d'éléments que présentent les candidats moins les éléments appartenant à leur préfixes communs. Alors que pour l'arbre de hachage, ce nombre est supérieur à celui que présentent les candidats au total. En effet il faut stocker les feuilles (donc les candidats, i.e. tous leurs éléments) et les chemins de l'arbre qui y conduisent.

– Une plus grande **rapidité de détection** des inclusions. Intuitivement, il suffit d'observer la figure 1.6 et de simuler la détection des candidats inclus dans la séquence de données < (20) (40) (20 30) > pour s'en rendre compte. En effet, avec cette séquence, l'algorithme va parcourir l'arbre uniquement à partir du sommet 20 (fils de la racine). Comme 10 ne fait pas partie de cette séquence, toute la partie gauche de l'arbre (i.e. en dessous du sommet 10, fils de la racine) sera ignorée. De plus, une structure de hachage aurait stocké les candidats qui commencent par 20 dans une même feuille. Arrivé sur cette feuille il aurait fallu, à nouveau, lancer une comparaison entre les séquences de cette feuille et la séquence de données qui y a conduit. Dans l'arbre préfixé, l'algorithme est guidé par la séquence jusqu'à ce que le parcours échoue, signe de non inclusion des séquences représentées par les feuilles qui sont descendantes du sommet de cet échec. Si le parcours réussit alors les séquences, représentées par les chemins de la racine aux feuilles sur lesquelles l'algorithme aboutit, sont des candidats inclus dans la séquence de données testée.

1.3 La généralisation des motifs séquentiels

Pour répondre à des besoins de type marketing, les auteurs de [SA96b] ont voulu améliorer la notion de motifs séquentiels, en la généralisant. Les motifs séquentiels généralisés sont présentés comme des motifs séquentiels qui font preuve d'élasticité ou de rigidité par rapport aux changements d'itemsets.

Cette aptitude des motifs séquentiels généralisés à manipuler les changements d'itemsets est définie en fonction des contraintes de temps. Il s'agit en fait de deux façons d'envisager le changement d'itemset :

- Un point de vue **restrictif**. Dans ce cadre, les changements d'itemset doivent respecter un certain écart de temps (i.e. être au delà ou bien en deça d'un écart fixé). Cette fonction permet à l'algorithme d'extraction de motifs séquentiels de trouver des comportements à court terme ou, au contraire, des comportements à long terme.
- Un point de vue **laxiste**. Dans ce cadre, le changement d'itemset peut être occulté, pour faire une sorte d'essai ("et si les achats du 10 et du 11 Juillet étaient considérés comme simultanés, cela ferait-il apparaître un nouveau fréquent ? "). Cette fonction permet donc d'extrapoler les données de la base, en procédant à de légères modifications, afin d'en connaître les conséquences et observer l'éventuelle apparition de nouveaux fréquents.

1.3.1 Les contraintes de temps

Les contraintes de temps ont pour but de modifier le critère d'inclusion entre un candidat et la séquence de données en cours d'évaluation. Leur but est de considérer les moyens possibles de réfuter une inclusion ou, au contraire, de contraindre une inclusion qui ne devrait pas l'être. Dans ce but, les contraintes de temps sont au nombre de trois :

- minGap : dont le but est d'estimer si deux itemsets sont trop proches pour être considérés comme appartenant à la même séquence.
- maxGap : dont le but est, au contraire, d'estimer si deux itemsets sont trop éloignés pour être considérés comme appartenant à la même séquence.
- windowSize : dont le but est de fusionner deux itemsets pour créer un itemset virtuel qui pourrait autoriser l'inclusion.

Les exemples 4 et 5 illustrent ces concepts, de façon intuitive tout d'abord, puis de façon plus formelle.

Client	01/10/2001	02/10/2001	15/10/2001
Client 1	PC 1Ghz Windows 2000		Livre « Linux Facile »
Client 2	PC 1Ghz	Windows 2000	Livre « Linux Facile »

FIG. 1.7 – *Une base de données exemple*

Exemple 4 *La figure 1.7 représente les achats de deux clients dans un supermarché. Un processus d'extraction de motifs séquentiels sur cette base, avec un support minimum de 100%, fera apparaître deux comportements fréquents. Le premier de ces comportements est "100% des clients achètent dans l'ordre, un* PC 1Ghz *et, à une date ultérieure, un livre* <Linux Facile>*". Le second comportement fréquent est "100% des clients achètent dans l'ordre, un logiciel* Windows 2000 *et, à une date ultérieure, un livre* <Linux Facile>*". Considérons à présent que le propriétaire de cette base désire connaître les effets d'une fusion entre les journées successives. Cela revient à grouper deux à deux les journées*

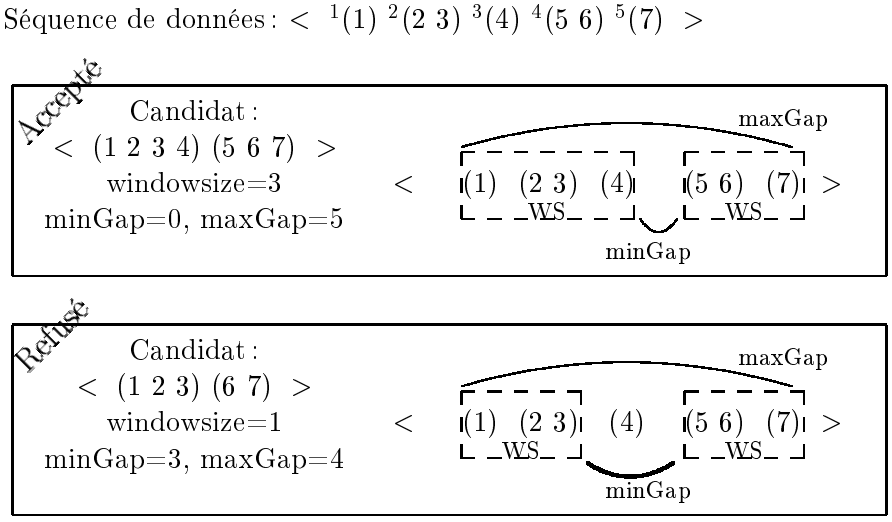

FIG. 1.8 – *Illustration du fonctionnement des contraintes de temps*

d'achat pour considérer qu'elles n'en font qu'une. Par exemple les journées du 01 et 02 Septembre 2001 seront groupées pour le client 2 afin de tester cette configuration et observer les résultats. Un nouveau comportement est, en effet, issu de cette fusion : "100% des clients achètent simultanément un PC 1Ghz *et un logiciel* Windows 2000. *Cet achat est suivi, à une date ultérieure, par l'achat d'un livre* <Linux Facile>".

Exemple 5 *La figure 1.8 détaille un peu plus le fonctionnement des contraintes de temps. Considérons que le processus d'extraction soit en train d'évaluer les candidats inclus dans (supportés par) la séquence de données d suivante :* $< {}^{1}(1)\ {}^{2}(2\ 3)\ {}^{3}(4)\ {}^{4}(5\ 6)\ {}^{5}(7) >$ *(dans laquelle chaque itemset est estampillé par sa date, par exemple l'itemset* (2 3) *a été acheté à la date* 2*). Pour déterminer les candidats inclus dans cette séquence, l'algorithme doit tester les configurations possibles de d car les contraintes de temps autorisent un grand nombre de combinaisons. Prenons l'exemple de la séquence* < (1 2 3 4) (5 6 7) > *(premier candidat de la figure). En temps normal ce candidat ne devrait pas être considéré comme supporté par d. Pourtant, en jouant sur la contrainte de temps windowSize, les trois premières journées d'achats (itemsets) de d peuvent être regroupées pour n'en former qu'une. Cet itemset virtuel permet alors au candidat d'être considéré comme inclus dans la séquence d. Notons également que les contraintes minGap et maxGap sont respectées. Considérons, à présent, la séquence* < (1 2 3) (6 7) >. *Ce candidat n'est pas, en temps normal, supporté par d. En jouant sur la contrainte windowSize, on peut regrouper les deux premiers et les deux derniers itemsets de d. Cependant, malgré cette modification, on réalise que minGap n'est pas respecté entre les deux itemsets crées et le candidat n'est donc pas inclus dans la séquence de données d.*

Évaluation des contraintes à la volée

Les contraintes de temps sont apparues avec [SA96b], dans lequel les auteurs proposent l'algorithme GSP, destiné à les prendre en compte. GSP se base sur la structure de hachage déjà proposée dans [AS95]. Pour évaluer les contraintes de temps au cours d'une passe, GSP va calculer, à la volée, toutes les combinaisons possibles de la séquence de données évaluée, pour chaque sommet de l'arbre. Cette méthode présente, à notre sens, deux problèmes majeurs :

- Des **combinaisons recalculées** à chaque étape. En effet, les combinaisons dues aux contraintes de temps sont nombreuses, mais elles restent les mêmes d'un sommet à l'autre.
- Des **combinaisons inutiles**. La réflexion que nous avons menée sur les contraintes de temps nous a conduit vers la conclusion qu'un certain nombre d'entre elles suffisait à couvrir toutes les autres.

1.3.2 Contribution

Notre étude vise donc à répondre aux deux problèmes rencontrés lors de l'évaluation des contraintes de temps à la volée. Pour les résoudre nous allons proposer un graphe, adapté à la problématique des contraintes de temps, qui représente les combinaisons possibles. Ce graphe sera donc calculé une fois pour toutes et les combinaisons qu'il représente seront utilisées à chaque sommet de l'arbre. De plus, une optimisation sur les inclusions entre les combinaisons nous permet de réduire le nombre de celles-ci, de manière significative. L'exemple 6 propose un survol de la méthode GTC que nous proposons pour résoudre le problème des contraintes de temps ([MPT99a, Mas98]).

Exemple 6 *Avec la base de données représentée par la figure 1.9, si windowSize vaut 5 et minGap vaut 1, les combinaisons à tester sont les suivantes :*

$< (1) (2) (3) (5) (6) > \bullet$
$< (1) (2) (3) (4) (6) > \bullet$
$< (1) (2) (3) (4\ 5) (6) > \circ$
$< (1) (2) (3\ 4) (6) > \cdot$
$< (1) (2) (3\ 4\ 5) (6) > \odot$

On peut alors noter que les séquences marquées par un $\bullet$ *sont incluses dans celle marquée par un* $\circ$*, alors que celle marquée par un* $\cdot$ *est incluse dans celle marquée par un* $\odot$*. Les seules combinaisons à tester sont alors :*

$< (1) (2) (3) (4\ 5) (6) >$
$< (1) (2) (3\ 4\ 5) (6) >$

En effet, on remarque que si un candidat est supporté grâce à la combinaison $< (1) (2) (3) (5) (6) >$ *alors ce candidat est forcément supporté par la combinaison* $< (1) (2) (3) (4\ 5) (6) >$*. Le graphe de la figure 1.10 représente alors les combinaisons pertinentes qu'il faut tester pour cette séquence de données. En effet, dans ce graphe, tout chemin qui part d'un sommet source et se termine par un sommet puits représente une séquence à tester.*

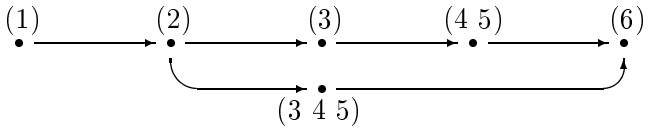

FIG. 1.10 – *Un graphe de séquence, calculé avec windowSize*

Client	Date	Item
C1	01/04/98	1
C1	07/04/98	2
C1	13/04/98	3
C1	17/04/98	4
C1	18/04/98	5
C1	24/04/98	6

FIG. 1.9 – *Exemple de base de données*

Nous verrons, de plus, comment les travaux que nous avons menés sur GTC permettent à cet algorithme de se combiner avec les parcours de structures de candidats. Un des avantages de GTC est en effet de pouvoir s'adapter aux structures qui gèrent les candidats, comme par exemple l'arbre préfixé de PSP ou encore l'arbre de hachage utilisé par GSP.

1.4 L'extraction incrémentale de motifs séquentiels

Parce que les bases de données traitées sont régulièrement mises à jour, le besoin d'une approche incrémentale semble évident. De plus, si l'on considère que les séquences fréquentes que l'on va découvrir, après la mise à jour, sont une modification des séquences découvertes avant cette mise à jour, alors la nécessité de cette approche n'est plus à démontrer. Les bénéfices, que l'on peut tirer d'un raisonnement incrémental sur la fouille de données, sont largement exploités pour les règles d'association [CHNW96, CLK97, AP95, SS98, TBAR97, RMR96, RMR97]. La problématique des motifs séquentiels se doit également de s'adapter au problème des mises à jour et de l'incrémentalité.

1.4.1 Intérêts d'une approche incrémentale

La rapidité avec laquelle sont stockées les données implique, du point de vue de la fouille de données, un contexte très différent du contexte statique. En effet, prendre en compte l'obsolescence des anciens motifs découverts, peut également nous faire réfléchir à la façon de les utiliser pour découvrir les plus récents. Ce principe permet alors de procéder à une nouvelle étape de fouille de données sur la base qui exploite la mise à jour et les anciens fréquents, afin d'en extraire les nouveaux. L'exemple 7 illustre la problématique de l'extraction incrémentale de motifs séquentiels.

Exemple 7 *La base de données DB figure 1.11 représente les achats de 4 clients, et couvre un période de trois jours. La précédente étape de fouille de données sur DB, avec un support de 50%, a permis*

Client	Itemsets		
$C1$	10 20	20	50 70
$C2$	10 20	30	40
$C3$	10 20	40	30
$C4$	60	90	

(DB)

Itemsets	
50 60 70	*80 100*
50 60	*80 90*

(db)

FIG. 1.11 – *Une base de données (DB) et l'incrément (db) contenant de nouvelles transactions*

de trouver deux séquences fréquentes : $<(10\ 20)\ (30)>$ *et* $<(10\ 20)\ (40)>$. *Considèrons désormais la mise à jour effectuée sur DB, représentée par db à la figure 1.11. L'objectif est alors de trouver les nouveaux fréquents sur* $DB + db$, *en exploitant les connaissances apportées par la précédente phase de fouille. L'algorithme doit alors aboutir à la découverte de* $<(60)\ (90)>$, $<(10\ 20)\ (50\ 70)>$, $<(10\ 20)\ (30)\ (50\ 60)\ (80)>$ *et* $<(10\ 20)\ (40)\ (50\ 60)\ (80)>$ *(toujours avec un support de 50%).*

Dans l'exemple 7, nous pouvons noter la présence de $<(10\ 20)>$ (qui était déjà fréquent) dans trois des nouvelles séquences, sur quatre. Il est facile d'en déduire l'intérêt qu'une méthode de fouille de données aurait à tirer profit de cette information avant d'extraire les nouvelles séquences. Cela justifie, de manière intuitive, les avantages d'une méthode incrémentale par rapport à un processus de fouille de données classique, sur la totalité de la base (après la mise à jour), qui ne tiendrait pas compte de cette connaissance.

Méthode naïve pour l'extraction incrémentale

Une première méthode à laquelle nous pourrions penser pour exploiter les connaissances acquises, lors de la précédente fouille de données, consiste à tester les concaténations entre les séquences de DB et celles de db. Plus précisemment, soit $Freq_{DB}$ les fréquents découverts sur DB et soit $Freq_{db}$ les fréquents qui apparaissent sur db, cette méthode applique l'algorithme suivant :

$\forall f_1 \subseteq Freq_{DB}$
  $\forall f_1 \subseteq Freq_{db}$
    compterSupport$(f_1 + f_2)$

Cet algorithme tente, en quelque sorte, de construire les ponts qu'empruntent les fréquents qui sont partagés entre DB et db. Considèrons en effet la séquence à découvrir $<(10\ 20)\ (50\ 70)>$, qui est fréquente après la mise à jour de la base de données de la figure 1.11. D'après notre algoritme naïf, $<(10\ 20) \subseteq Freq_{DB}$ et $(50\ 70)> \subseteq Freq_{db}$, donc le candidat $<(10\ 20)(50\ 70)>$ sera construit, testé et validé. Mais cette méthode présente une complexité que l'on ne peut admettre, en raison du nombre de candidats à tester (fonction compterSupport) sur U (avec U=$DB + db$). Ce nombre est minimisé par la formule $NF_{DB} \times (k!) \times NF_{db} \times (j!)$, avec :

- NF_{DB} : le nombre de fréquents sur DB (i.e. avant la mise à jour).
- k : la longueur des fréquents sur DB.

- NF_{db} : le nombre de fréquents qui apparaissent sur *db*.
- j : la longueur des fréquents qui apparaissent sur *db*.

Si l'on instancie cette formule avec les valeurs 300 pour NF_{DB}, 60 pour NF_{db}, 8 pour k et 6 pour j, qui sont des valeurs très faibles pour un algorithme de fouille de données à l'heure actuelle, alors on obtient 10^9 candidats à tester. Dans la mesure où il est préférable d'obtenir les résultats de ce processus avant la prochaine mise à jour, une optimisation apparaît comme évidente.

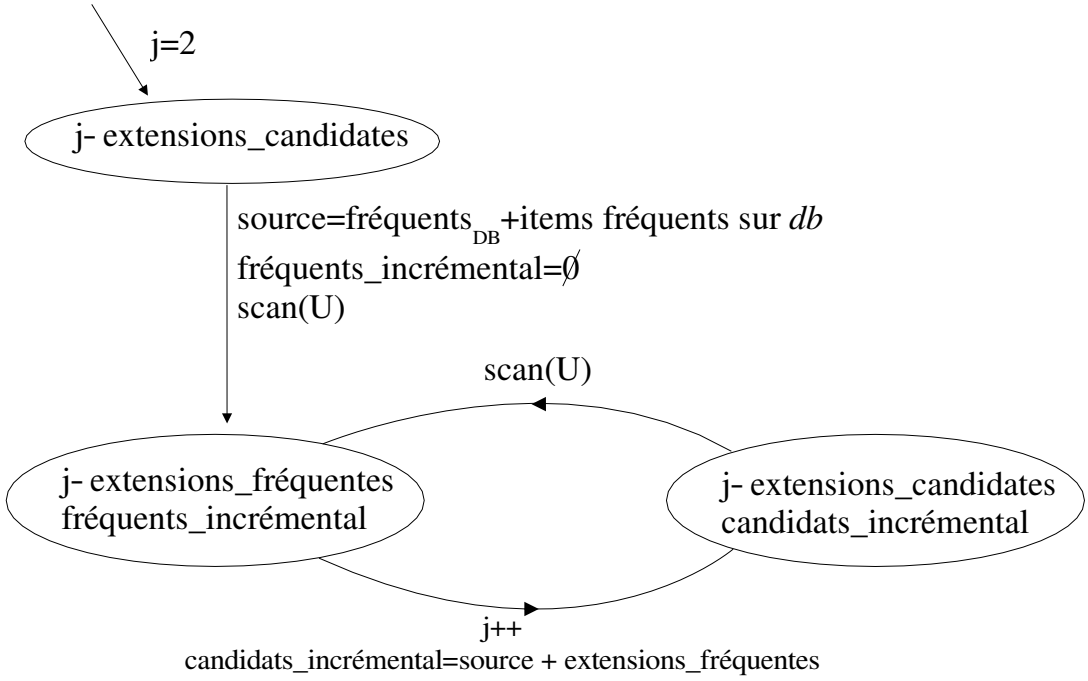

FIG. 1.12 – *L'automate d'extraction, version incrémentale*

1.4.2 Contribution

Le principe général de l'algorithme ISE, que nous avons mis en place ([MPT99b, MPT00]) est résumé par la figure 1.12. Il consiste en une reconception de la méthode générer-élaguer. La première étape de cet algorithme cherche les items fréquents qui appartiennent à *db*. Il s'agit là de la partie du "pont" (pour reprendre l'image donnée un peu plus haut), entre DB et *db*, qui se trouve du côté de *db*. Ensuite l'algorithme va se servir de cette information pour construire un ensemble de base, appelé source. Cet ensemble est constitué des fréquents de DB auxquels on a ajouté un item fréquent sur *db*. Si la séquence qui résulte de cette opération est fréquente sur U alors elle est ajoutée à *source*. Une

passe sur U permet alors de déterminer les extensions fréquentes de taille 2 et les séquences appartenant à *source*. À partir des extensions fréquentes de taille j, on construit les extensions candidates de taille $j+1$. De plus, a partir des extensions fréquentes de taille j et de l'ensemble source, on construit les candidats à l'incrémental. Une passe sur U permet de déterminer les nouvelles extensions fréquentes et les nouveaux fréquents découverts de manière incrémentale. Ce principe est également décrit, d'un point de vue itératif cette fois, par la figure 1.13.

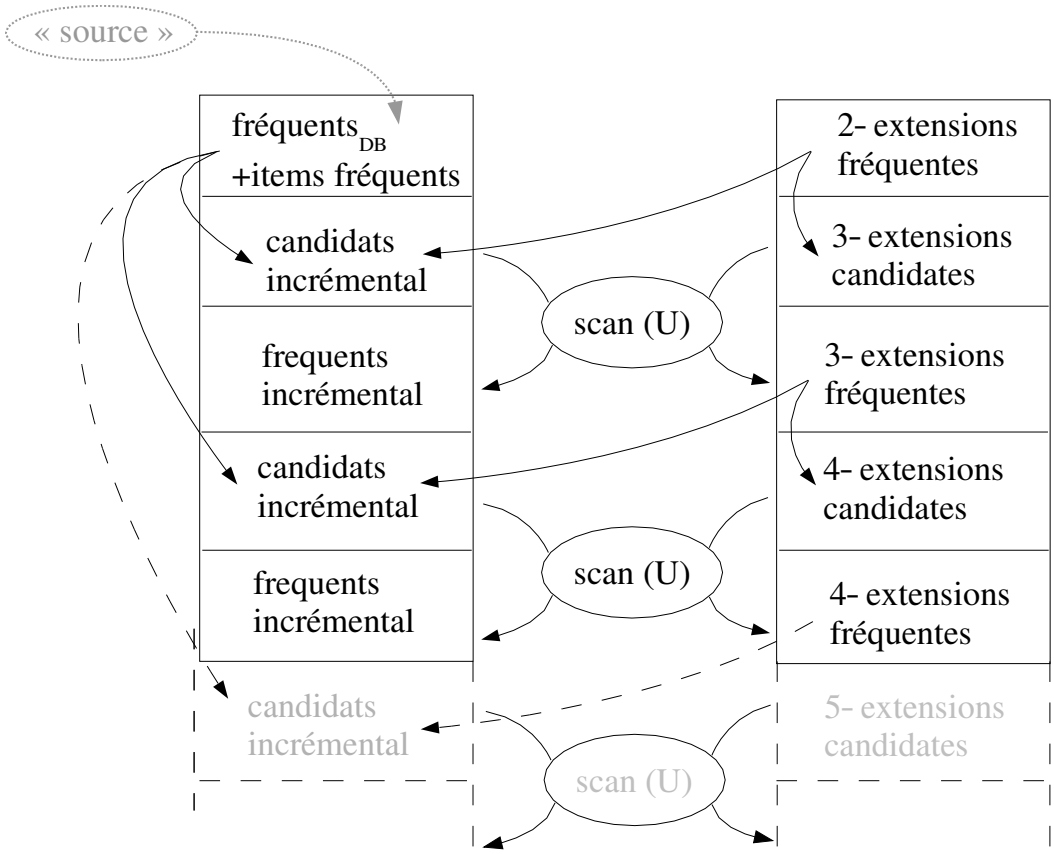

FIG. 1.13 – *Point de vue itératif sur* ISE

Nous avons également mis en place un certain nombre d'optimisations sur ISE. Ces dernières ont une efficacité suffisante pour réduire le nombre de candidats à tester de façon très efficace. Le nombre de candidats est en fait assez faible pour permettre de procéder à une étape de fouille de données plus rapide que l'exécution de GSP sur U, même sans avoir procédé à une étape de fouille sur DB. Plus précisément, nous avons constaté qu'il était plus efficace d'enchaîner les opérations suivantes, plutôt que d'utiliser GSP sur U :

- Découper la base à traiter en DB+db
- Utiliser GSP sur DB
- Utiliser ISE sur U

Le temps total T_{ISE} nécessaire est alors inférieur au temps T_{GSP}, nécessaire à GSP sur U. Toutefois T_{ISE} reste bien sûr supérieur à $T_{ISE_{db}}$, le temps nécessaire pour utiliser ISE sur U, sans avoir à découper la base et en considérant que GSP a déjà été utilisé sur DB.

1.5 Schema mining

Ces dernières années, les Systèmes de Gestion de Bases de Données (SGBD) ont considérablement évolué pour prendre en compte les besoins de nouvelles applications. Par exemple, un grand nombre de travaux ont été réalisés pour définir des systèmes capables de manipuler des données qui ne sont pas en première forme normale. Les extensions, proposées notamment par les systèmes objet, et même à l'heure actuelle par les entrepôts de données, permettent ainsi de stocker des structures de plus en plus complexes.
Malheureusement les approches de fouilles de données actuelles ne prennent pas en compte de telles structures et se contentent d'analyser des structures "plates", i.e. des éléments atomiques comme les achats de clients par exemple. Pourtant, dans de nombreuses applications (i.e. les "workflows", les systèmes actifs, les systèmes de gestion de contraintes, les systèmes transactionnels, les applications fonctionnant sur des SGBD Orientés Objets, etc.), la recherche de régularités dans des transactions imbriquées peut permettre d'obtenir de nombreuses informations pour améliorer la qualité des applications, pour optimiser les accès, pour favoriser les techniques de rétro-conception, pour assurer un meilleur suivi de l'exécution de tâches ou de méthodes, etc.

1.5.1 La structure commune à un ensemble de documents

Avec la popularité du World Wide Web (Web), le nombre de documents semi-structurés produits augmente très rapidement. Alors que dans les applications de bases de données classiques, nous décrivons d'abord la structure des données, i.e. le type ou le schéma, et nous créons les instances de ces types, dans les données semi-structurées, les données n'ont pas de schéma prédéfini et chaque objet contient sa propre structure [Wor97].
En effet, la majorité des documents "en-ligne", tels que les fichiers HTML, Latex, Bibtex ou SGML sont semi-structurés. Par conséquent, la structure des objets est irrégulière et il est judicieux de penser qu'une requête sur la structure des documents est aussi importante qu'une requête sur les données [WL99]. Cependant, cette irrégularité structurelle n'implique pas qu'il n'existe pas de similitudes structurelles parmi les objets semi-structurés. Il est même parfois assez fréquent de constater que des objets semi-structurés qui décrivent le même type d'informations ont des structures similaires. L'analyse de telles régularités dans des données semi-structurées peut alors fournir des informations très utiles pour le concepteur ou l'utilisateur d'un site.

1.5.2 Contribution

Nous proposons de procéder à un survol des techniques que nous avons mises en œuvre ([LMP00, LMPT00]), par le biais d'un exemple. La figure 1.14 représente un des types de données sur lesquels

nous avons travaillé, dans le cadre du Schéma Mining, ainsi que le résultat recherché. Il s'agit de la structure des pages HTML qui recensent les informations sur les membres d'une association, dans l'intranet de cette dernière. Pour cet exemple, les fiches sont au nombre de deux, représentées dans les cadres A et B de la figure 1.14.

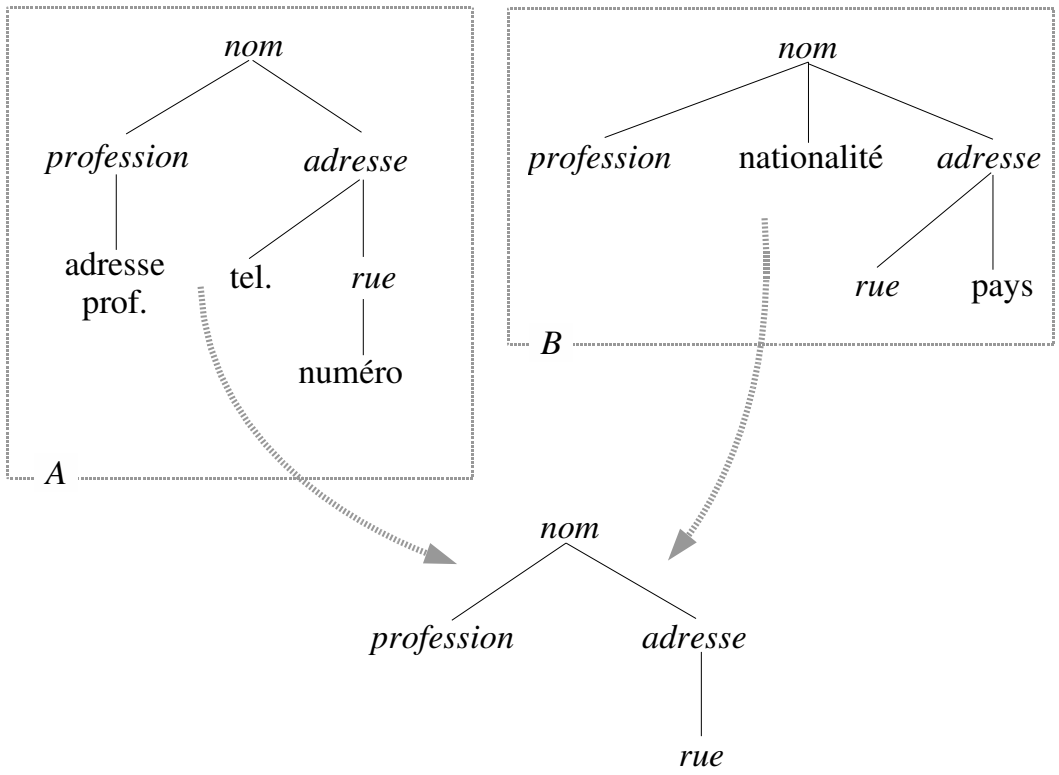

FIG. 1.14 – *Types de données pour le Schéma Mining*

Le but de l'extraction de schéma sera alors de trouver une structure commune à ces deux fiches, représentée par le troisième schéma de la figure 1.14. En effet, on peut constater que ces deux figures ont en commun une structure de base : le *nom* du sociétaire, sa *profession* et son *adresse* (qui est composée du seul nom de la rue). Ensuite, les informations disponibles varient, d'une fiche à l'autre.

Pour procéder à la recherche d'une éventuelle structure commune à ces deux schémas, nous proposons de passer par les étapes suivantes :

- Transformation des données. Dans cette étape nous attribuons à chaque élément (sommet) des schémas, un identifiant associé à sa profondeur dans l'arbre.
- Réduction au problème de l'extraction de motifs séquentiels. Le but, à ce stade de la méthode d'extraction, est de créer des séquences qui représentent les schémas.

- Une fois ces séquences créées, il est alors possible de procéder à une phase d'extraction de motifs séquentiels. Cette phase aboutit sur la découverte de séquences communes aux séquences qui représentent les schémas.
- Ces séquences communes sont, elles aussi, la représentation de schémas. Ces schémas, obtenus par une transformation inverse de celle effectuée lors de la réduction, sont alors des structures communes aux schémas traités.

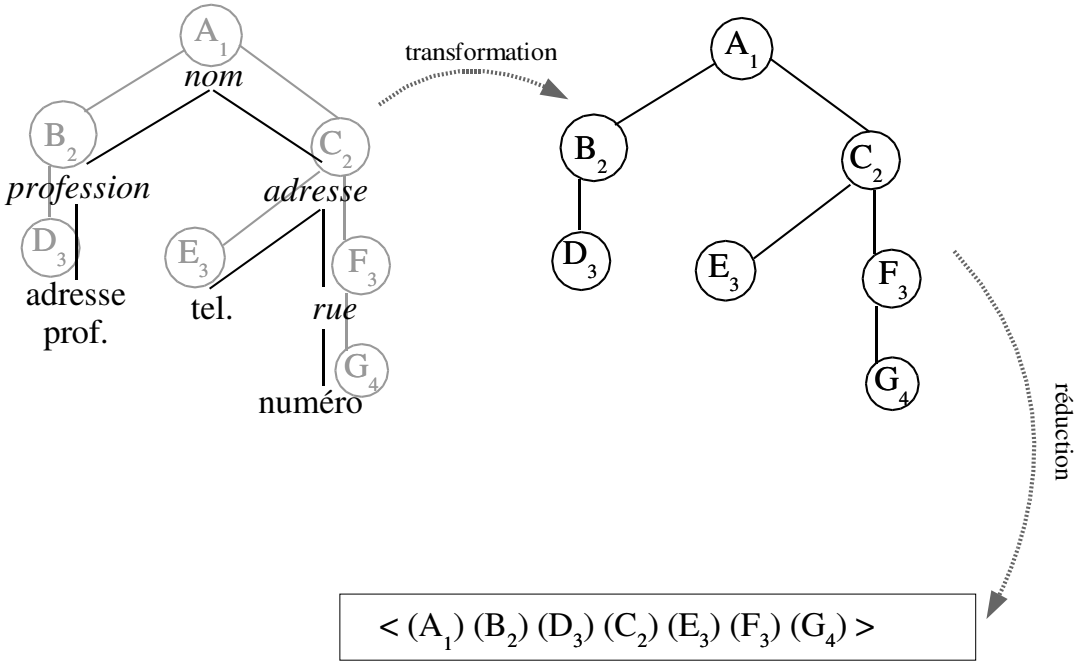

FIG. 1.15 – *Phases de transformation et de réduction pour le Schéma Mining*

Pour revenir à l'exemple de la figure 1.14, la transformation des données est décrite par la figure 1.15. Le sommet *nom* est alors traduit par l'identifiant "A". Comme ce sommet est la racine de l'arbre, il est indexé par son niveau : 1. Les autres sommets sont alors traités de la même façon. La séquence du schéma est obtenue par un parcours préfixé de l'arbre, après transformation. Pour l'arbre du cadre *A* (figure 1.14) la séquence obtenue est $< (A_1)\ (B_2)\ (D_3)\ (C_2)\ (E_3)\ (F_3)\ (G_4) >$. La séquence du second schéma, quant à elle sera : $< (A_1)\ (B_2)\ (G_2)\ (C_2)\ (F_3)\ (I_3) >$. La phase d'extraction de motifs séquentiels sur ces séquences, avec un support de 100%, permet d'obtenir la séquence commune : $< (A_1)\ (B_2)\ (C_2)\ (F_3) >$. La retranscription de cette séquence en schéma (rédution inversée suivie de transformation inversée) se solde alors par l'obtention du troisième schéma de la figure 1.14.

1.6 Web Usage Mining

Avec la popularité du World Wide Web, de très grandes quantités de données comme l'adresse des utilisateurs ou les URL demandées sont automatiquement récupérées par les serveurs Web et stockées

dans des fichiers access log. L'analyse de tels fichiers peut offrir des informations très utiles pour, par exemple, améliorer les performances, restructurer un site ou même cibler le comportement des clients dans le cadre du commerce électronique.
La découverte d'informations à partir du Web est généralement appelée Web Mining et peut recouvrir deux aspects : *Web content Mining* et *Web usage Mining* [CMS97]. La première approche concerne la découverte et l'organisation d'informations extraites du Web. Par exemple, des approches basées sur les agents sont utilisées pour découvrir et organiser, de manière autonome, les informations extraites à partir du Web [Lie95, KMA+98, MG98, PMB96] et des approches Bases de Données s'intéressent aux techniques d'intégration, d'organisation et d'interrogation de données hétérogènes et semi-structurées sur le Web [AQM+97, MAG+97, CGMH+94, FFKL98]. Le *Web usage Mining* s'intéresse par contre au problème de la recherche de motifs comportementaux des utilisateurs à partir d'un ou plusieurs serveur Web afin d'extraire des relations entre les données stockées.

1.6.1 L'analyse du comportement des utilisateurs d'un site

L'activité des utilisateurs d'un site web est enregistrée de façon systématique, sous forme de fichiers[8]. Ces enregistrements se présentent sous la forme décrite par la figure 1.16, qui illustre un extrait de fichier log.

```
132.208.12.150 - - [29/Nov/1998:18:02:30 +0200] "GET /info/RECRUT.gif HTTP/1.0" 200 1141
132.208.12.150 - - [29/Oct/1998:18:03:07 +0200] "GET /info/recrut.html HTTP/1.0" 200 1051
132.208.127.200 - - [16/Oct/1998:20:34:32 +0100] "GET /geaaix/home.html HTTP/1.0" 200 14617
148.241.148.34 - - [31/Oct/1998:01:17:40 +0200] "GET /info/index.html HTTP/1.0" 304 -
148.241.148.34 - - [31/Oct/1998:01:17:42 +0200] "GET /info/recrut.html HTTP/1.0" 304 -
192.70.76.73 - - [22/Nov/1998:11:06:11 +0200] "GET /info/program.html HTTP/1.0" 200 4280
192.70.76.73 - - [22/Nov/1998:11:06:12 +0200] "GET /info/MATIERES.gif HTTP/1.0" 200 2002
```

FIG. 1.16 – *Exemple de fichier access log*

On remarque, dans les enregistrements rapportés par la figure 1.16, que les informations disponibles sont principalement l'adresse IP de la machine utilisée pour naviguer, la date de l'action et l'URL demandée. Accessoirement, d'autres informations sont présentes, comme le protocole, le sens du flux ("get" ou "post"), etc.

L'analyse du comportement des utilisateurs d'un site web, a donc pour but de déterminer l'existence d'enchaînements fréquents parmi les séquences de navigation des utilisateurs. Si une séquence de navigation est commune à n utilisateurs, avec n le support minimum fixé par le propriétaire du site, alors cette découverte peut conduire à un certain nombre de conclusions. Une de celles-ci peut consister à remodeler son site en fonction des navigations les plus fréquentes. Nous verrons par la suite que la re-conception du site, en fonction des motifs fréquents dans le fichier log, peut se faire de façon statique ou bien de façon dynamique.

8. Le fichier contenant l'activité des utilisateurs d'un site web sera désigné par "log" ou "access log".

1.6.2 Contribution

Notre contribution relative au Web Usage Mining se décompose, dans un premier temps, en deux parties ([MPC99a, MPC99b, MPC99c, MPT99c, MPC00]). La première concerne l'extraction de navigations fréquentes sur le site alors que la seconde concerne l'exploitation que l'on peut faire de ces découvertes.

Extraction de navigations fréquentes

Les similarités entre le fichier log et l'enregistrement des achats qu'ont effectué les clients dans un supermarché sont nombreuses. Il est en effet facile d'assimiler le numéro de machine à un numéro de client, la date du fichier log à une date classique et enfin l'URL demandée à un article (ou item). Une fois cette transposition faite, il reste encore à grouper les enregistrements par client. Le format de données obtenu est alors parfaitement adapté pour un algorithme d'extraction de motifs séquentiels, tel que Psp. Les résultats obtenus peuvent alors subir une transformation inverse, afin de fournir des résultats compréhensibles par le propriétaire du site.

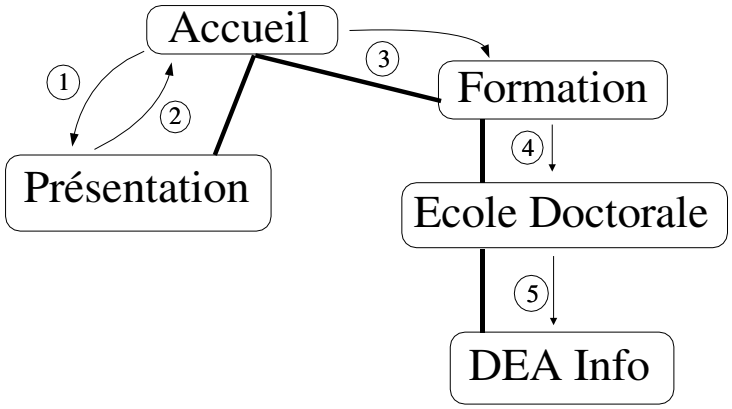

Fig. 1.17 – *Exemple de navigation fréquente*

Ces étapes (transformation, extraction de motifs séquentiels, transformation inverse) nous ont conduit à la découverte de séquences de navigation fréquentes, comme celle décrite par la figure 1.17. Le résultat exprimé par cette figure, nous indique un comportement fréquent, de la part des utilisateurs du site web du LIRMM. Ce comportement consiste à commencer la navigation par la page d'accueil, puis la page de présentation du laboratoire, avant un retour sur la page d'accueil. La navigation se poursuit par une visite de la page des formations, de l'école doctorale et enfin du DEA d'informatique.

Exploitation des résultats

Parmi les utilisations que l'on peut faire des connaissances acquises par un processus de Web Usage Mining, figure la modification du site concerné. Cette modification peut se faire de manière statique

(i.e. prendre en compte les comportements fréquents les plus significatifs et modifier définitivement le site) ou de manière dynamique. La modification dynamique, que nous avons mise en place, repose sur une architecture illustrée par la figure 1.18.

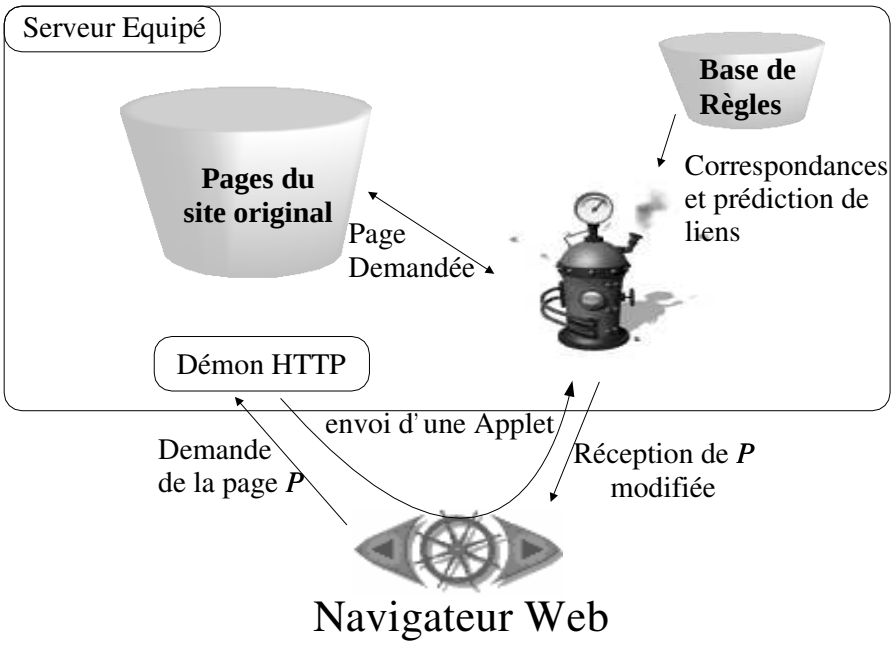

FIG. 1.18 – *Architecture de la modification dynamique de site web*

Le principe de cette architecture peut se décomposer en plusieurs étapes. Un utilisateur u demande la page p sur le site. La demande est prise en compte par le serveur HTTP, qui renvoie à l'utilisateur une page contenant une applet. Cette applet est alors chargée de rentrer en communication avec le moteur de prédiction. Ce moteur consulte la base des règles comportementales et tente d'apparier le comportement de u avec un des comportements fréquents. Le moteur va ensuite chercher la page p demandée par u. Si le comportement de u peut être utilisé avec un comportement fréquent pour prévoir la suite des opérations de u, alors p sera modifiée en fonction de la prédiction qui peut être faite. Sinon p sera envoyée à u sans modification. Parmi les modifications que p peut subir, nous pouvons citer, par exemple, l'ajout de liens vers les pages qui font partie de la prédiction pour la navigation de u.

1.7 L'efficacité d'une approche inter-sites en Web Usage Mining

Une des conséquences majeures de la popularité grandissante du web, s'est traduite par l'apparition puis le développement d'un phénomène de liens commerciaux, permettant la navigation de site en site. Ces liens, que nous pouvons qualifier de partenariat, ouvrent alors de nouvelles perspectives sur la façon de voir le comportement des utilisateurs.

1.7.1 Les comportements dans le contexte du partenariat

Dans le contexte du partenariat, il ne s'agit plus d'analyser le comportement des utilisateurs d'un site, mais d'analyser le comportement des utilisateurs communs à plusieurs sites. Nous proposons une nouvelle problématique pour prendre en compte ce facteur incontournable : le Web Usage Mining Inter-sites. Nous verrons comment les comportements découverts prennent une dimension supplémentaire, en raison de leur projection multi-sites.

Prenons le bref exemple d'un site de vente en ligne SV, qui propose des articles très variés. Ces articles viennent de fabricants qui possèdent, chacun, un site sur le Web : $SF1$, $SF2$, ..., SFn. Chacun de ces sites propose la description de ses produits (entre autres informations) et des liens vers les sites de revendeurs proposant ses articles (dont SV). Le visiteur d'un site SFx a alors de fortes chances d'être redirigé vers SV s'il décide de passer à l'achat. Pour SV, tenir compte de l'historique de ce visiteur et des comportements fréquents de ceux qui, avant lui, ont suivi cette redirection, présente sans nul doute des intérêts que la seule analyse du fichier access log de SV ne suffit pas à satisfaire.

1.7.2 Contribution

Nous proposons, dans un premier temps, d'apporter une dimension supplémentaire au Web Usage Mining classique (Intra-Site). Dans un deuxième temps, nous présentons une étude visant à mettre en corrélation les propriétés des fréquents, à l'intersection des données mises en jeu, et le support minimum.

La problématique

Le premier aspect de notre contribution propose de définir le concept de Web Usage Mining Inter-Sites. Dans ce cadre, les motifs recherchés ont une portée plus grande en comparaison au Web Usage Mining Intra-Site. Ils permettent de découvrir des comportements fréquents pour une population ciblée, correspondant aux visiteurs communs à deux sites Web. Le fait de restreindre la phase d'extraction de connaissances sur cette partie de la population répond à un objectif déterminant du processus d'ECD : la sélection, étape du prétraitement des données. Du point de vue du Web Usage Mining Inter-Sites, les visiteurs sont naturellement indentifiés comme appartenant à la catégorie que l'on va étudier. Dans notre exemple, cette catégorie est constituée de l'intersection entre l'ensemble des visiteurs du site SFx d'une part, et ceux de SV d'autre part.

Pour le site SV, qui veut connaître le comportement des visiteurs que lui envoie $SF1$, les motifs séquentiels découverts dans le cadre du Web Usage Mining Inter-Sites sont alors de la forme suivante :

70 % des clients envoyés par SF1, ont visité, de façon séquentielle, les pages suivantes :
`SF1(/téléviseurs/home-cinéma/cat1/modèles.html)`
`SF1(/téléviseurs/home-cinéma/cat1/modèle3/revendeur1.html)`
`SV(/magasin/hi-fi/home-cinéma/SF1/tarif-modèle3.html)`
`SV(/magasin/hi-fi/home-cinéma/SF1/tarif-modèle4.html)`

```
SV(/magasin/hi-fi/enceintes/SF1/modeles.html)
SV(/magasin/hi-fi/enceintes/SF3/tarif-modèle8.html)
```

Les avantages se situent alors :

- dans le potentiel offert par ces résultats : si SV veut modifier l'organisation hypertexte pour un client venu de SF1 et dont le comportement est similaire à ce comportement fréquent, alors cette modification peut prendre effet dès l'arrivée de ce client sur le site SV (contrairement aux techniques similaires dans le cadre du Web Usage Mining Intra-Site).
- dans la confiance avec laquelle ces résultats sont calculés : en effet tous les visiteurs ne sont pas pris en compte dans la phase d'extraction, donc les résultats obtenus sont dédiés aux visiteurs de la catégorie qui préoccupe SV (i.e. leur confiance est supérieure).

Le deuxième aspect de notre contribution prouve que la phase d'extraction de connaissances, mettant en jeu l'intersection des données provenant des deux sites, ne doit pas nécessairement être envisagée comme un processus classique. En effet, les propriétés que nous utilisons nous permettent de conclure sur la possibilité d'employer un algorithme de fouille de données incrémental, dont les gains, en termes de temps de calcul, ont été mis en relief dans [PZOD99, CLK97, CHNW96, LL98, MPT99b].

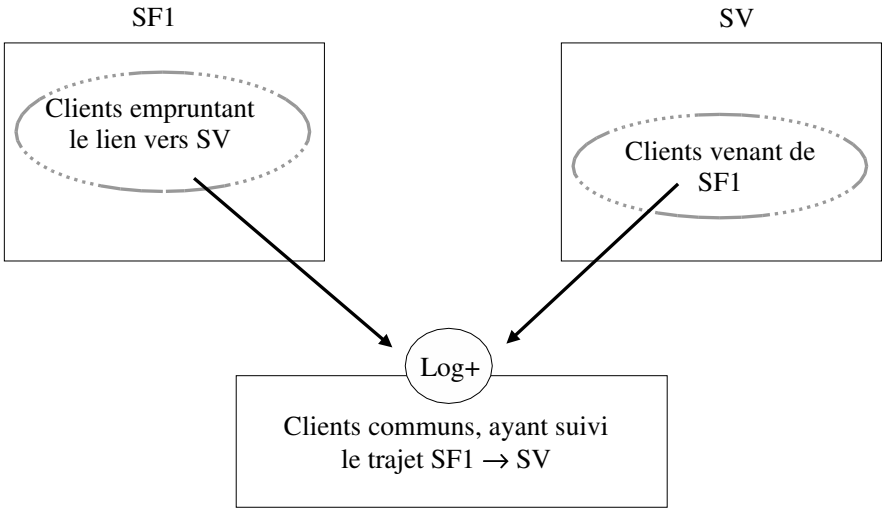

FIG. 1.19 – *Mise en commun des fichiers access log*

Méthode proposée

La mise en commun des fichiers log des sites A et B, illustrée par la figure 1.19, montre une analogie singulière entre la conception du log correspondant au site virtuel $A \cup B$ d'un côté et la mise à jour d'une base données, dans le cadre de l'incrémentalité, de l'autre. Nous allons donc exploiter cette analogie, en faisant du Web Usage Mining Inter-Sites une instance de l'extraction incrémentale de

motifs séquentiels. Nous pourrions en effet considérer que la partie du log de $A \cup B$ qui se limite aux navigations sur A sera assimilée à DB et que la partie qui se limite aux navigations sur B sera assimilée à *db*. Le facteur manquant reste les fréquents sur DB (i.e. A). Nous considérons que les sites mis en jeu par le Web Usage Mining Inter-Sites sont à même de fournir les comportements fréquents de leurs utilisateurs. De plus nous considérons que les sites engagés dans le cadre d'un partenariat sont prêts à partager les navigations fréquentes découvertes avec leurs partenaires. Désignons par F_{log_A} les comportements fréquents sur le site de A, calculés avec un support s. Notre étude prouve que les fréquents sur A avec pour support n peuvent servir de base pour une extraction incrémentale sur le log de $A \cup B$, avec une support m supérieur à n et dépendant du nombre de clients dans chacun des deux fichiers. Une fois cette étude prouvée ([MTP01b]), nous mettrons en place une adaptation de la problématique de l'extraction incrémentale de motifs séquentiels, destinée à résoudre le problème du Web Usage Mining Inter-Sites.

1.8 Une solution temps réel pour le Web Usage Mining

Les solutions présentées jusqu'ici pour le Web Usage Mining ont en commun un facteur qui peut paraître pénalisant selon le contexte : la vitesse à laquelle les comportement varient. En effet, sur le web, les changements de comportement sont facilités par divers facteurs, comme par exemple le caractère arborescent des liens hypertexte. Si un site présente des comportements très changeants chez ses utilisateurs, alors ce site n'aura probablement pas d'intérêt à utiliser les comportements fréquents découverts la veille, voire même quelques jours auparavant. Si la fouille de données est, à l'image de la lanterne de Confucius, un moyen de mieux connaître son passé, nous estimons qu'analyser avec rapidité son passé le plus proche, le plus récent, constitue le meilleur moyen d'appréhender un futur tout aussi proche, à savoir les événements à venir dans l'immédiat. C'est dans cet optique que nous proposons d'étudier les possibilités relatives à une méthode de Web Usage Mining en temps réel.

1.8.1 Les enjeux d'une solution temps réel

Les concepts relatifs à l'analyse du comportement des utilisateurs d'un site web ont pour objectifs la prévision des demandes, la réorganisation du site ou encore l'optimisation des performances réseaux. Les analyses évoquées seront basées sur une fouille *a posteriori* des informations collectées sur les actions que les utilisateurs suivent sur un site : le fichier log. Cependant, si l'on considère la vitesse à laquelle les comportements des utilisateurs peuvent varier, le résultat obtenu risque de perdre de sa pertinence s'il est exploité trop tard. Dans le but de contourner cette obsolescence nous proposons HDM, un module de fouille de données distribué, basé sur une heuristique et conçu pour obtenir les motifs de comportements fréquents pour répondre au problème du Web Usage Mining en temps réel.

De plus, nous avons pour objectif de fournir, grâce à cette méthode, des résultats plus fins que ceux apportés par une méthode de Web Usage Mining classique :

- A l'heure actuelle, les méthodes classiques de Web Usage Mining offrent des motifs de comportements qui peuvent prendre place sur des jours, des semaines, voire des mois (en fonction de l'intervalle de temps sur lequel le fichier log est enregistré). Pour cette raison, si un motif de comportement est "caché" dans la masse de données considérée, mais si ce motif n'est fréquent que

lorsque l'on considère un intervalle bien précis et non le fichier dans son intégralité, alors ce motif fréquent ne sera pas trouvé par une méthode classique. Considérons le motif $F1$, qui apparaît pour dix clients entre le 12 et le 14 Juin, et nulle part ailleurs dans le fichier *Log* qui contient des enregistrements allant du 01 Mai au 30 Juillet. Considérons que du 12 au 14 Juin seuls 15 clients sont enregistrés dans le fichier *Log*, alors que ce même fichier contient en tout 100 clients. Finalement considérons que le support minimum fixé par l'utilisateur soit de 60%. Avec un tel support minimum, si l'on observe *Log* dans son intégralité, alors $F1$ ne peut pas être considéré comme fréquent car il n'apparaît que dans les séquences de 10 clients (soit un support de 10%). Cependant, le propriétaire du site Web serait certainement intéressé d'apprendre que du 12 au 14 Juin, 60% des clients ont eu un comportement particulier (à savoir $F1$). Une méthode classique d'analyse de fichier log ne pourrait faire apparaître un tel résultat sans traitement particulier (traitement qui, à notre connaissance, n'existe pas à l'heure actuelle).

- Une fois les résultats d'un processus de Web Usage Mining obtenus, les motifs de comportements fréquents sont pris en compte pour, entre autres, tenter de faire des prédictions. Quand plusieurs types de règles sont applicables pour un utilisateur u, elles sont toutes considérées à un niveau de valeur équivalent. Admettons qu'une de ces règles soit trouvée grâce aux navigations d'utilisateurs que l'on peut classer dans une catégorie identique à celle de u (des utilisateurs proches de u). La règle ainsi produite devrait surclasser les autres règles au moment de prédire le comportement de u. La question est alors de savoir quels utilisateurs peuvent être considérés comme "proches" des utilisateurs dont on veut prévoir le comportement. Dans la mesure où les utilisateurs dont on veut prévoir le comportement sont des utilisateurs connectés, une réponse que nous considérons pertinente est : les autres utilisateurs connectés. Puisque nous pouvons connaître les comportements fréquents des utilisateurs connectés en temps réel (grâce à notre proposition), nous sommes en mesure d'appliquer ce point de vue.

- La grande majorité des algorithmes de Web Usage Mining basés sur une analyse de fichier log sera bien entendu limitée par un aspect bien connu du problème : la taille du fichier à traiter en entrée. HDM est conçu pour fournir des résultats en temps réel, quel que soit le nombre de clients traités.

- Comme nous le verrons dans ce mémoire, plusieurs méthodes récentes sont destinées à répondre à un problème majeur de la recherche d'itemsets fréquents (ou de motifs séquentiels fréquents) : la taille des fréquents à trouver est tellement importante que la plupart des méthodes énumèrent un trop grand nombre de solutions intermédiaires avant de saturer la mémoire disponible. HDM est conçu pour faire face à ce problème également, en fournissant des résultats satisfaisants, quelle que soit la taille des fréquents à obtenir.

- Enfin, notre proposition repose sur le fait que, quand un utilisateur est connecté à un site Web, la machine qui permet à son navigateur de fonctionner fournit une puissance de calcul qui, à elle seule, peut paraître faible, mais envisagée par centaines (voire par milliers) devient plus que suffisante, à condition de savoir l'exploiter.

Dans la mesure où notre proposition est basée sur une heuristique, notre but est de fournir un résultat répondant aux critères suivants :

Soit $resultatReel$ le résultat à obtenir (le résultat qu'obtiendrait un algorithme de fouille de données qui explore tout l'ensemble des solutions après analyse du fichier Log). $resultatReel$ est alors l'ensemble des motifs séquentiels à trouver. Soit $resultatHDM$ les résultats obtenus par la méthode proposée dans ce mémoire. Nous voulons minimiser $\sum_{i=0}^{taille(resultatHDM)} S_i/S_i \notin resultatReel$ tout en maximisant $\sum_{i=0}^{taille(resultatReel)} R_i/R_i \subseteq resultatHDM$.
En d'autres termes, nous voulons trouver toutes les séquences apparaissant dans $resultatReel$ tout en évitant que le résultat soit plus grand qu'il ne le devrait (sinon l'ensemble de toutes les séquences, de toutes les navigations, pourrait constituer un résultat, car il englobe le résultat réel).

1.8.2 Contribution

Le principe, sur lequel se base notre méthode ([MTP02, MTP01a]), consiste à envisager la machine de l'utilisateur non seulement comme une machine connectée mais aussi comme une puissance de calcul disponible. Il est possible de l'exploiter si l'on conçoit une architecture adéquate. Cette puissance de calcul ne représente pas un bénéfice considérable à l'unité, mais elle devient largement suffisante si elle est multipliée par autant d'utilisateurs connectés.

Notre but est donc d'exploiter un peu de la puissance de calcul des machines connectées, pour implémenter une heuristique distribuée de Web Usage Mining. Cette heuristique emprunte aux algorithmes génétiques leur conception du voisinage, tout en gardant les propriétés des motifs fréquents à l'esprit pour optimiser les candidats proposés. Les générations de candidats sont alors testées par les machines utilisateurs, afin que chacune compare les candidats avec sa propre séquence de navigation. Cette comparaison a pour but de noter les candidats, et la somme des notes permet de connaître les candidats les plus "côtés". La figure 1.20 donne un aperçu de l'architecture mise en place.

1.9 LeitmotiV : un cadre de travail pour le processus d'ECD

Le nombre d'applications sur lesquelles nous avons concentré nos efforts, nous a conduit vers le développement d'une plateforme d'intégration de nos algorithmes. Elle a pour but de centraliser les méthodes d'extraction de motifs séquentiels, afin d'accepter différents types de données en entrée et de fournir des résultats uniformisés à l'utilisateur. Cette plateforme, appelée *L*eitmotiV [9], est à l'heure actuelle capable de traiter les étapes suivantes :

- **Prétraitement** des données à l'aide d'un parser existant.
- Acquisition des **paramètres** de l'étape fouille de données.
- Procéder à une **fouille de données**.
- La transformation inverse (processus opposé du prétraitement) des résultats afin d'obtenir une **sortie lisible** par l'humain, grâce à un deparser,
- **Manipulation**, stockage, et interrogation des résultats.

9. Les tarvaux menés dans le cadre de la réalisation de *L*eitmotiV ont donné lieu à un projet de type CPER et un projet de type RNTL.

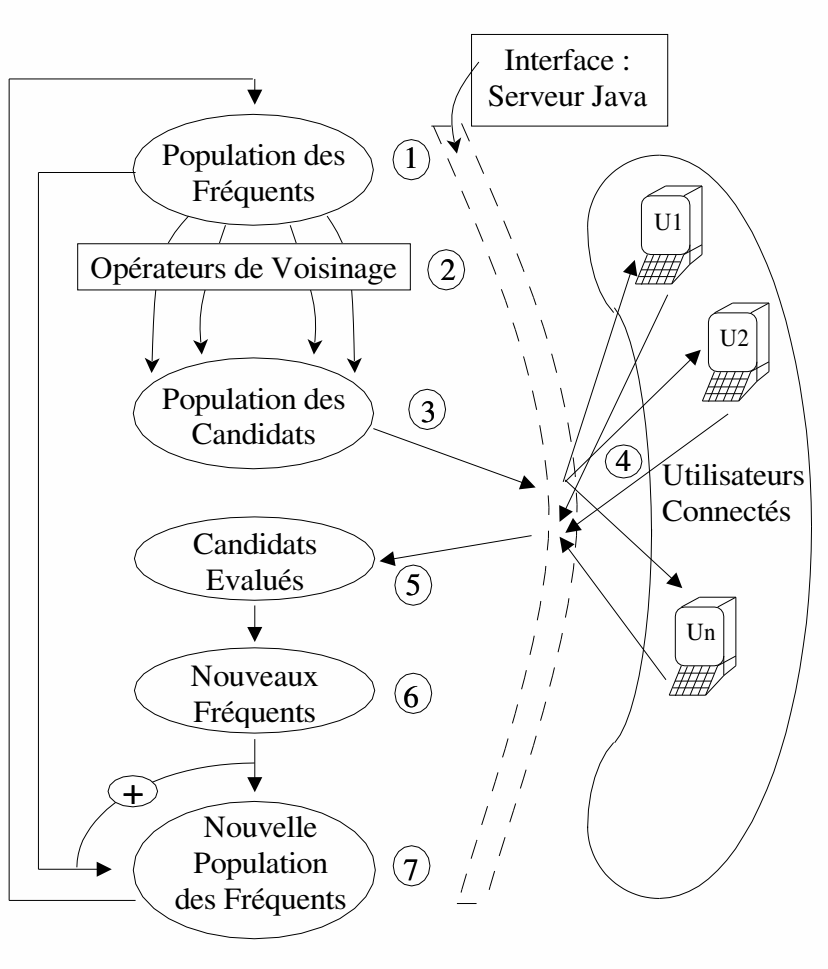

FIG. 1.20 – *Vue d'ensemble de l'heuristique distribuée*

Ces étapes sont généralement les maillons d'une chaîne qui va de l'acquisition des données jusqu'à la production et la présentation des résultats d'un algorithme de fouille de données. Cependant, quelle que soit l'application, une étape d'intérêt majeur reste le prétraitement des données. Ce prétraitement est à l'heure actuelle assuré par des parsers qui, à partir des données en entrée, vont fournir un format de données lisibles par l'algorithme de fouille. Cette transformation se fait selon des critères précis, déterminés par le responsable de la fouille, qui devra trouver la meilleure re-écriture des données de base. De la qualité de cette re-écriture dépend la qualité des résultats et donc la qualité de tout le processus d'ECD. Nous estimons, cependant, que l'étude d'une automatisation de cette étape (ou de certaines parties de cette étape) peut s'avérer bénéfique.
Dans cette optique, une phase de subsitution, destinée à remplacer la transformation des données, fait à l'heure actuelle l'objet de travaux au sein de l'équipe Data Mining du Lirmm. Cette étape consiste à automatiser la traduction des données en proposant de composer elle-même la transformation (parser) à employer, plutôt que d'utiliser un parser existant.

1.10 Organisation du mémoire

Les concepts présentés dans ce mémoire s'articulent autour de la présentation suivante :

La première partie de ce mémoire regroupe la présentation des problèmatiques et l'état de l'art qui leur est relatif. Le chapitre 2 présente les problématiques que nous traitons. Nous y verrons les définitions de quelques problèmes directement liés à l'extraction de motifs séquentiels. Ce chapitre présente aussi les définitions nécessaires pour aborder les applications dérivées de cette problématique. Nous y présentons les bases du Schema Mining et du Web Usage Mining. Ce dernier thème fera l'objet d'une présentation détaillée car nous y voyons les problématiques liées à l'inter-sites et à l'analyse du comportement d'utilisateurs en temps réel. Le chapitre 3 propose un état de l'art des techniques liées aux méthodes que nous proposons. Nous y étudions les algorithmes et architectures qui nous semblent les plus proches ou les plus reconnus dans les domaines sur lesquels nous travaillons.

La deuxième partie présente les algorithmes que nous avons developpés pour l'extraction de motifs séquentiels, ainsi que quelques applications. Notre première contribution est détaillée dans le chapitre 4. Nous y expliquons le fonctionnement de notre structure de hachage, ainsi que l'algorithme Psp destiné à l'exploiter. Le chapitre 5, présente l'algorithme Gtc, conçu pour étendre Psp. Cette extension passe par la prise en compte des contraintes de temps et les optimisations que l'on peut implémenter pour les traiter. Ise, l'algorithme incrémental pour la recherche de motifs séquentiels, est expliqué par le chapitre 6. Enfin, les méthodes qui permettent le Text Mining et le Schema Mining sont détaillées dans le chapitre 7.

La troisième partie est destinée à mettre en relief un domaine particulièrement bien adapté à la problématique des motifs séquentiels : le Web Usage Mining. Nous verrons, tout d'abord, une architecture destinée à analyser le comportement des utilisateurs d'un site web dans le chapitre 8. Notre algorithme Wumis pour le Web Usage Mining Inter-Sites est décrit dans le chapitre 9 et, enfin, l'architecture Hdm pour le Web Usage Mining temps réel est détaillé dans le chapitre 10.

Dans la dernière partie, les chapitres 11 et 12 proposent les conclusions que nous pouvons tirer de nos travaux, ainsi que les perspectives qu'ils offrent.

Première partie

Problématique et état de l'art

Chapitre 2

Problématiques

Nous proposons, dans ce chapitre, d'examiner les différentes problématiques abordées dans ce mémoire. La problématique de la recherche des *règles d'association* y est abordée en premier lieu (à la section 2.1), car les principes et notions qu'elle requiert seront utilisés tout au long de ce document. Les solutions algorithmiques et les méthodes que nous apportons, sont articulées autour de la recherche des *motifs séquentiels* fréquents. La problématique de cette recherche est donc expliquée en détails à la section 2.2, avant que leur *aspect généralisé* soit présenté à la section 2.3. La *recherche incrémentale de motifs séquentiels* manipule les concepts définis plus tôt, mais dans un contexte différent et avec des contraintes supplémentaires. Ce sont ce contexte et ces contraintes que nous décrivons à la section 2.4.

Les applications qui peuvent être faites, à partir des problématiques présentées, sont alors assez riches pour devenir elles-mêmes des domaines à part entière. Nous avons abordé, parmi les utilisations possibles des motifs séquentiels, le *Schema Mining*, dont une présentation est faite à la section 2.5. Le *Web Usage Mining* lui se décline en plusieurs problématiques corrélées mais distinctes. L'analyse du comportement des utilisateurs d'un site web est décrite à la section 2.6. À partir de cette problématique, nous avons constaté que le concept de Web Usage Mining pouvait être amélioré dans sa définition. Nous avons donc proposé de définir la problématique du *Web Usage Mining Inter-sites* à la section 2.7, avant de proposer un concept *temps réel*, dont les objectifs sont décrits à la section 2.8.

2.1 Recherche de règles d'association

Formellement, le problème des règles d'associations est présenté dans [AIS93] de la façon suivante : soit $I = \{i_1, i_2, \ldots, i_m\}$ un ensemble d'items. Soit DB un ensemble de transactions, tel que chaque transaction T, dotée d'un identifiant unique noté (TID), soit constituée d'un un ensemble d'items, appelés *itemset* vérifiant $T \subseteq D$. Une transaction T supporte X, un ensemble d'items de I, s'il contient tous les items de X, i.e. si $X \subseteq T$. Une règle d'association est une implication de la forme $X \rightarrow Y$, où $X \subseteq I$, $Y \subseteq I$ et $X \cap Y = \emptyset$. La confiance c dans la règle établit la probabilité qu'une transaction T supportant l'item X, supporte aussi l'item Y. Le support d'une règle $X \rightarrow Y$ correspond au pourcentage de toutes les transactions de DB supportant $X \cup Y$.
Ainsi, à partir d'une base de transaction DB, le problème consiste à extraire toutes les règles d'association dont le support est supérieur à $minSupp$ et dont la confiance est supérieure à $minConf$ (confiance spécifiée par l'utilisateur). Cette tâche peut être divisée en deux étapes :

1. Rechercher dans un premier temps tous les itemsets fréquents. Si la base possède m items, 2^m items sont alors potentiellement fréquents, d'où le besoin de méthodes efficaces pour limiter cette recherche exponentielle.
2. Générer, à partir de ces itemsets fréquents, des règles dont la confiance est supérieure à $minConf$. Cette étape est relativement simple dans la mesure où il s'agit de conserver les règles du type $A/B \rightarrow B$ (avec $B \subset A$), ayant un taux de confiance suffisant.

2.2 Recherche de motifs séquentiels

La recherche de motifs séquentiels (ou séquences) est à la base de la recherche de motifs séquentiels généralisés (ou séquences généralisées) qui seront définie à la section 2.3. C'est pourquoi nous reprenons, dans la section 2.2.1, les définitions et propriétés qui nous seront utiles tout au long de ce document.

2.2.1 Définitions et propriétés

Le problème de la recherche de séquences dans une base de données de transactions est présenté dans [AS95] de la façon suivante (nous gardons, au niveau des définitions, les concepts de clients et d'achats, cependant nous adapterons par la suite la recherche de séquence à d'autres types de bases de données) :

Définition 1 Une *transaction*[1] constitue, pour un client C, l'ensemble des items achetés par C à une même date. Dans une base de données client, une transaction s'écrit sous la forme d'un ensemble : id-client, id-date, itemset. Un *itemset* est un ensemble d'items non vide noté $(i_1 i_2 \dots i_k)$ où i_j est un *item* (il s'agit de la représentation d'une transaction non datée). Une *séquence* est une liste ordonnée, non vide, d'itemsets notée $< s_1 s_2 \dots s_n >$ où s_j est un itemset (une séquence est donc une suite de transactions qui apporte une relation d'ordre entre les transactions). Une *séquence de données* est une séquence représentant les achats d'un client. Soit $T_1, T_2, \dots, T_n$ les transactions d'un client, ordonnées par dates d'achat croissantes et soit itemset(T_i) l'ensemble des items correspondants à T_i, alors la séquence de données de ce client est $< itemset(T_1)\ itemset(T_2) \dots\ itemset(T_n) >$.

Exemple 8 *soit C un client et S=< (3) (4 5) (8) >, la séquence de données représentant les achats de ce client. S peut être interprétée par "C a acheté l'item 3, puis en même temps les items 4 et 5 et enfin l'item 8".*

Définition 2 Soit $s_1 =< a_1 a_2 \dots a_n >$ et $s_2 =< b_1 b_2 \dots b_m >$ deux séquences de données. s_1 est *incluse* dans s_2 ($s_1 \prec s_2$) si et seulement si il existe $i_1 < i_2 < \dots < i_n$ des entiers tels que $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}, \dots\ a_n \subseteq b_{i_n}$.

Exemple 9 *La séquence $s1$ =< (3) (4 5) (8) > est incluse dans la séquence $s2$ =< (7) (3 8) (9) (4 5 6) (8) > (i.e $s1 \prec s2$) car* (3) $\subseteq$ (3 8), (4 5) $\subseteq$ (4 5 6) *et* (8) $\subseteq$ (8). *En revanche* $<$ (3) (5) $> \not\prec <$ (3 5) $>$ *(et vice versa).*

Définition 3 Un client *supporte* une séquence s (fait partie du support pour 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 des 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*.

Remarque 1 *Une séquence de données n'est prise en compte qu'une seule fois pour calculer le support d'une séquence fréquente, i.e. elle peut présenter plusieurs fois le même comportement, le processus de recherche de séquences considère qu'elle produit ce comportement sans tenir compte du nombre de ses apparitions dans la séquence de données.*

1. Nous considérons ici le terme de transaction dans le sens d'une transaction financière et non pas dans celui d'une transaction dans une base de données.

Client	Date	Items
C_1	01/01/1998	20,60
C_1	02/02/1998	20
C_1	04/02/1998	30
C_1	18/02/1998	80,90
C_2	11/01/1998	10
C_2	12/01/1998	30
C_2	29/01/1998	40,60,70
C_3	05/01/1998	30,50,70
C_3	12/02/1998	10,20
C_4	06/02/1998	20,30
C_4	07/02/1998	40,70
C_4	08/02/1998	90

FIG. 2.1 – *Base de donnés exemple*

Définition 4 Soit une base de données DB, l'ensemble L^{DB} des *séquences fréquentes maximales* (également notées *motifs séquentiels*) est constitué de toutes les séquences fréquentes telles que pour chaque s dans L^{DB}, s n'est incluse dans aucune autre séquence de L^{DB}.

Le problème de la recherche de séquences maximales (*sequential patterns* dans [AS95]) consiste à trouver l'ensemble L^{DB} de la définition 4.
Les deux propriétés suivantes considèrent le cas des sous-ensembles par rapport aux calculs du support et de l'inclusion.

propriété 1 *Soit s_1 et s_2 deux séquences. Si $s_1 \prec s_2$ alors $support(s_1) \geq support(s_2)$.*

propriété 2 *Soit s_1 une séquence non fréquente. Quelle que soit s_2 telle que $s1 \prec s_2$, s_2 est une séquence non fréquente.*

La propriété 1 se justifie par le fait que toute séquence de données d dans DB supportant s_2 supporte obligatoirement s_1 (l'inverse ne se vérifie pas). La propriété 2 est une conséquence de la propriété 1. En effet, d'après cette propriété, $support(B) \leq support(A) < support_{min}$, donc B n'est pas fréquent.

2.2.2 Exemple

Considérons la base de données DB illustrée par la figure 2.1. Avec un support minimum de 50% (i.e. pour qu'une séquence soit retenue, il faut que deux clients dans la base de données supportent cette séquence), les séquences fréquentes sont alors les suivantes : < (20) (90) >, < (30) (90) > et < (30) (40,70) >. Les deux premières font partie du support pour C_1 et C_4, alors que la dernière apparaît dans les séquence de données des clients C_2 et C_4.

2.3 Motifs séquentiels généralisés

Le problème de la recherche de séquences, défini dans la section 2.2, présente des limites qui ont conduit au problème de la recherche de séquences généralisées. Ces limites, auxquelles se proposent de répondre les algorithmes de recherche de séquences généralisées, sont les suivantes :

- **Absence de contraintes de temps**. Souvent, l'application peut avoir besoin de séparer les transactions pour ne pas prendre en compte des événements trop éloignés ou trop rapprochés (selon le but, on peut vouloir réaliser par exemple une étude statistique à court terme ou bien à long terme).
- **Rigidité de la définition des transactions**. Nombre d'applications peuvent avoir besoin de ne pas tenir compte d'une très faible durée séparant deux transactions et considérer qu'elles ont eu lieu, non pas de manière séquentielle, mais simultanée, comme une seule et même transaction.

2.3.1 Définitions des contraintes de temps

Le problème de la recherche de séquences généralisées est défini dans [SA96b] de la façon suivante :

Définition 5 Soient *minGap* une durée minimum, *maxGap* une durée maximum et *windowSize* une durée de rectification (également notée ws), spécifiées par l'utilisateur. Une séquence de données $d =< d_1...d_m >$ supporte une séquence $s =< s_1...s_n >$ si il existe des entiers $l_1 \leq u_1 < l_2 \leq u_2 < ... < l_n \leq u_n$ tels que:

1. s_i est incluse dans $\cup_{k=l_i}^{u_i} d_k$, $1 \leq i \leq n$;
2. date(d_{u_i}) - date(d_{l_i}) $\leq windowSize$, $1 \leq i \leq n$;
3. date(d_{l_i}) - date($d_{u_{i-1}}$) $> min\text{-}gap$, $2 \leq i \leq n$;
4. date(d_{u_i}) - date($d_{l_{i-1}}$) $\leq max\text{-}gap$, $2 \leq i \leq n$.

Le *support* de s, noté $supp(s)$, est le pourcentage de toutes les séquences dans DB qui supportent s. Si $supp(s) \geq \sigma$, avec une valeur de support minimum σ, la séquence s est dite *fréquente*.

Ce concept est destiné à rendre les dates associées aux transactions des clients plus "maléables", dans la mesure où il est désormais possible de :

- regrouper des achats lorsque leurs dates sont assez proches (cette opération est apportée par windowSize),
- considérer des itemsets (achats) comme trop rapprochés pour apparaître dans la même séquence fréquente (il s'agit de la contrainte de minGap),
- considérer des itemsets (achats) comme trop éloignés pour apparaître dans la même séquence fréquente (il s'agit de la contrainte de maxGap).

L'exemple 10 est destiné à comprendre le fonctionnement des contraintes de temps appliquées à la recherche de séquences.

Exemple 10 *Pour illustrer le fonctionnement des contraintes de temps, considérons une base de données réduite à un client :*

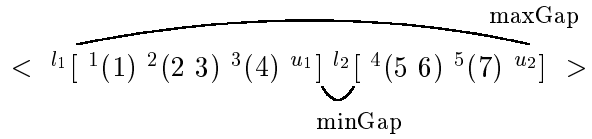

FIG. 2.2 – *Illustration des contraintes de temps*

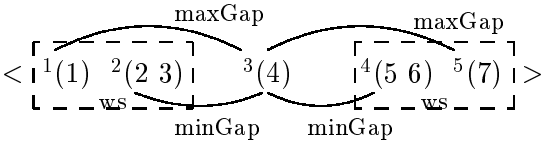

FIG. 2.3 – *Illustration des contraintes de temps*

Customer	Time	Items
C1	01/01/1998	1
C1	02/02/1998	2,3
C1	03/02/1998	4
C1	04/02/1998	5,6
C1	05/02/1998	7

La séquence de données d correspondant au client $C1$ est alors la suivante :
$d =<\ ^{1}(1)\ ^{2}(2\ 3)\ ^{3}(4)\ ^{4}(5\ 6)\ ^{5}(7)\ >$ où chaque itemset est estampillé par son jour d'achat, i.e. $^{4}(5\ 6)$ signifie que les items 5 et 6 ont été achetés le 4/02/1998.

Considérons une séquence candidate c, nous illustrons cinq possibilités différentes d'affectations de valeurs pour c, windowSize, minGap et maxGap afin d'illustrer la signification des contraintes de temps.

c=< (1 2 3 4) (5 6 7) >, windowSize=3, minGap=0 et maxGap=5

La séquence candidate est alors considérée comme incluse dans la séquence de données pour deux raisons :

1. le paramètre windowSize permet de regrouper les itemsets (1) (2 3) et (4) ainsi que les itemsets (5 6) et (7) afin d'obtenir les itemsets (1 2 3 4) et (5 6 7),
2. la contrainte minGap entre les itemsets (4) et (5 6) est respectée.

Pour reprendre les paramètres de la définition donnée dans la section 2.3.1, nous avons $l_1 = 1$, $u_1 = 3$, $l_2 = 4$, $u_2 = 5$ et la séquence de données d est alors manipulée comme illustré par la figure 2.2.

c=< (1 2 3) (4) (5 6 7) >, windowSize=1, minGap=0 et maxGap=2

La séquence candidate est alors considérée comme incluse dans la séquence de données. En effet, nous avons $l_1 = 1$, $u_1 = 2$, $l_2 = 3$, $u_2 = 3$, $l_3 = 4$ *et* $u_3 = 5$. La figure 2.3 illustre la prise en compte des contraintes de temps pour la séquence de données d.

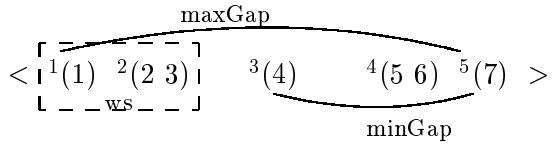

FIG. 2.4 – *Illustration des contraintes de temps*

Client	Date	Items
C_1	1	Ringworld
C_1	2	Foundation
C_1	15	Ringworld Engineers, Second foundation
C_2	1	Foundation, Ringworld
C_2	20	Foundation and Empire
C_2	50	Ringworld Engineers

FIG. 2.5 – *Base de données exemple*

c=< (1 2 3) (7) >, windowSize=1, minGap=3 et maxGap=4

La séquence candidate est alors considérée comme incluse dans la séquence de données. En effet nous avons $l_1 = 1$, $u_1 = 3$, $l_2 = 5$, $u_2 = 5$. La figure 2.4 illustre la prise en compte des contraintes de temps pour la séquence de données d.

c=< (1 2 3 4) (7) >, windowSize=1, minGap=3 et maxGap=4

Il n'y a aucun moyen de regrouper des itemsets de d de façon à accepter s comme séquence incluse. En effet windowSize=1 ne suffit pas à regrouper les itemsets (1) (2 3) et (4).

c=< (1 2 3) (6 7) >, windowSize=1, minGap=3 et maxGap=4

Dans ce cas, il n'y a qu'un seul moyen d'obtenir les deux itemsets (1 2 3) et (6 7) en utilisant windowSize pour regrouper les itemsets [(1) (2 3)] puis [(5 6) (7)] mais dans ce cas minGap n'est plus respecté entre les deux itemsets car ils sont espacés de seulement 2 jours ($date(2\ 3) = 2$ et $date(5\ 6) = 4$) alors que la contrainte minGap est fixée à 3 jours.

2.3.2 Exemple

Ce premier exemple est tiré de [SA96b], où la recherche de séquences généralisées est présentée :
Considérons les séquence de données illustrées par la figure 2.3.2, où les dates d'achat sont représentées par des entiers. Avec un support minimum supérieur à 50% (les deux séquence de données doivent supporter une séquence pour qu'elle soit fréquente), d'après les définitions données dans la section 2.2.1, les motifs séquentiels sont : <(Ringworld) (Ringworld Engineers)> et <(Foundation) (Ringworld Engineers)>.
Si nous considérons maintenant un windowSize de 7 jours, la séquence fréquente suivante apparaît :

Client	Itemsets		
$C1$	10 20	20	50 70
$C2$	10 20	30	40
$C3$	10 20	40	30
$C4$	60	90	

(DB)

Itemsets	
50 60 70	*80 100*
50 60	*80 90*

(db)

FIG. 2.6 – *Une base de données (DB) et l'incrément (db) contenant de nouvelles transactions*

<(Foundation, Ringworld) (Ringworld Engineers)>. L'ensemble des motifs séquentiels est alors le suivant : E=<(Foundation, Ringworld) (Ringworld Engineers)>, car nous avons <(Foundation) (Ringworld Engineers)>$\prec$<(Foundation, Ringworld) (Ringworld Engineers)> et <(Ringworld) (Ringworld Engineers)>$\prec$<(Foundation, Ringworld) (Ringworld Engineers)>.
Considérons à présent une contrainte de maxGap fixée à 30 jours, alors plus aucune des séquences trouvées précédemment n'est fréquente car C_2 ne supporte plus aucune d'entre elles.
Reprenons à présent l'exemple de la section 2.2.2 et considérons un windowSize de 2 jours. Dans ces conditions une nouvelle séquence fréquente apparaît : <(20 30) (90)> car elle est alors supportée par C_1 et C_4. Supposons à présent que l'utilisateur spécifie une contrainte de maxGap fixée à 15 jours. La séquence <(30) (40 70)> ne peut alors plus être considérée comme fréquente dans la mesure où la séquence de données du client C_2 affiche une durée de 17 jours entre les deux itemsets de cette séquence.

2.4 Extraction incrémentale de motifs séquentiels

2.4.1 Définition

Soit DB une base de données et *minSupp* le support minimum spécifié par l'utilisateur lors d'une phase de fouille de données sur DB. Soit *db* la base de données incrément, contenant de nouvelles transactions et de nouveaux clients, qui constitue la mise à jour effectuée sur DB après la phase de fouille de données décrite précédemment. Nous considérons que chaque transaction de *db* est triée par client et par date. Soit $U = DB \cup db$ la base de données résultant de la mise à jour (i.e. intégrant les clients et transactions de *db* ajoutés à ceux de DB).

Soit L^{DB} l'ensemble de sequences fréquentes découvertes sur DB. Le problème de l'**extraction incrémentale de motifs séquentiels** consiste à découvrir l'ensemble des séquences fréquentes (au sens de la définition 4) sur U, noté L^U, en respectant le support *minSupp* énoncé plus haut. De plus l'approche incrémentale doit utiliser les résultats de la phase de fouille de données précédente (i.e. L^{DB}) afin d'éviter de re-calculer une (ou certaines) partie(s) du résultat final.

Client	Itemsets		
$C1$	10 20	20	50 70
$C2$	10 20	30	40
$C3$	10 20	40	30
$C4$	60	90	
$C5$			

(DB)

Itemsets	
50 60 70	*80 100*
50 60	*80 90*
10 40	*70 80*

(db)

FIG. 2.7 – *Une base de données (DB) et son incrément constitué de nouvelles transactions et de nouveaux clients (db).*

2.4.2 Décomposition du problèmes et illustration

Tout d'abord, nous considérons une sous-partie du problème, limitée au seul ajout de transactions à des clients existants dans la base de données d'origine. Pour illustrer ce problème, considérons la base DB de la figure 2.6, dans laquelle sont reportées les actions de 4 clients. Chaque transaction est triée par date. Par exemple la séquence du client $C3$ est $<$ (10 20) (40) (30) $>$. Considérons le support minimum *minSupp* de valeur 50%. Selon la définition du support minimum, pour être considérée comme fréquente une séquence doit apparaître (être incluse) dans (les transactions de) au moins deux clients. L'ensemble des séquences fréquentes découvertes sur la base DB est alors : $L^{DB} = \{<$ (10 20) (30) $>$, $<$ (10 20) (40) $>\}$. Après quelques mises à jour, considérons l'incrément db (également décrit à la figure 2.6) contenant de nouvelles transactions ajoutées aux clients $C2$ et $C3$. Avec le même support minimum, les séquences $<$ (60) (90) $>$ et $<$ (10 20) (50 70) $>$ deviennent fréquentes sur U. Considérons la première de ces séquences, elle n'est pas fréquente sur DB dans la mesure où le support minimum ne serait pas respecté (cette séquence n'est supportée que par un client). En considérant l'incrément (et donc U) cette séquence devient fréquente car elle apparaît dans les séquence de données des clients $C3$ et $C4$. La séquence $<$ (10 20) $>$ était incluse dans les séquence de données des clients $C1$, $C2$ et $C3$ dans la base de données d'origine DB. En prenant en compte l'incrément db, la nouvelle séquence fréquente $<$ (10 20) (50 70) $>$ est découverte car elle est supportée par les clients $C1$ et $C2$.

Remarquons enfin la découverte de deux autres séquences fréquentes sur U : $<$ (10 20) (30) (50 60) (80) $>$ et $<$ (10 20) (40) (50 60) (80) $>$. Notons que $<$ (50 60) (80) $>$ est une séquence fréquente sur db. Après une analyse de U nous pouvons constater qu'il s'agit de l'extension de deux séquences fréquentes sur DB (i.e. $<$ (10 20) (30) $>$ et $<$ (10 20) (40) $>$).

Considérons à présent le problème dans son intégralité, avec la prise en compte dans db de nouvelles transactions mais aussi de nouveaux clients (figure 2.7). Dans cet exemple, 50% reste le support minimum. Pour être considérée comme fréquente, une séquence doit être supportée par au moins trois clients, dans la mesure où un nouveau client ($C5$) a été ajouté. Dans cette nouvelle configuration l'ensemble des séquences fréquentes sur DB (L^{DB}) devient : $\{<$ (10 20) $>\}$ car les séquences $<$ (10 20) (30) $>$ et $<$ (10 20) (40) $>$ ne sont supportées que par les clients $C2$ et $C3$. Cependant la séquence $<$ (10 20) (50) $>$ reste fréquente car elle apparaît dans les séquence de données des clients $C1$, $C2$ et $C3$. En prenant la mise en jour en considération, l'ensemble des séquences fréquentes dans

la base de données mise à jour devient $L^U = \{< (10\ 20)\ (50) >, < (10)\ (70) >, < (10)\ (80) >, < (40)\ (80) >, < (60) >\}$. Observons plus attentivement la séquence $< (10\ 20)\ (50) >$. Cette séquence pouvait être detectée dans le client $C1$ dans la base d'origine DB, mais ce support n'était pas suffisant pour la considérer comme fréquente. Néanmoins, comme l'item 50 devient fréquent grâce à la base de données incrément, cette séquence est supportée par les clients $C2$ et $C3$. De la même façon, la séquence $< (10)\ (70) >$ devient fréquente dans la mesure où, avec l'incrément, elle est supportée par les clients $C1$, $C2$ et le nouveau client $C5$.

2.5 Schema Mining : recherche de structures fréquentes

2.5.1 Transactions Imbriquées

Le problème de la recherche de régularités dans des transactions imbriquées est basé sur la recherche de régularités d'items complexes dans une base de transactions. Un item, dans notre contexte, est soit simple, soit complexe. Un item simple correspond à un objet élémentaire de l'application (nom de fonction, nom d'une tâche, etc.) Un item complexe quant à lui est composé d'autres items via des constructeurs "liste-de" ou "ensemble-de".

Exemple 11 *Pour illustrer les notions d'items simples et complexes, considérons une base de données contenant des transactions constituées à partir de bibliothèques de programme. Chaque programme est défini soit par une tâche*[2] *élémentaire (un item simple) soit par une succession de tâches élémentaires (item complexe). Si nous considérons maintenant, plus en détail, une tâche complexe, celle-ci peut être vue comme la suite d'un ensemble de tâches complexes qui elles-mêmes sont formées de tâches complexes jusqu'à obtenir une tâche élémentaire. Considérons quatre tâches élémentaires a, b, c et d. Considérons les tâches complexes suivantes : $A = \{C,a,b\}$, et $C =< d,c >$. La tâche complexe A est en fait réalisée de la manière suivante : $\{< d,c >, a, b\}$. En d'autres termes, la réalisation de la tâche A consiste à d'abord exécuter soit les tâches élémentaires d et c (dans cet ordre) puis après la tâche a et b (dans n'importe quel ordre), soit la tâche a et b (sans ordre) puis l'association des tâches d et c.* □

La problématique liée à la recherche de régularités dans des transactions imbriquées est définie de la manière suivante : soit DB une base de données de transactions imbriquées. Une transaction t contient un élément E si et seulement si $E \subseteq t$. Le support de E est défini comme le pourcentage de transactions dans DB contenant E : $supp(E) = \|\{t \in DB \mid E \subseteq t\}\| / \|\{t \in DB\}\|$. Etant donné un support minimal, $minSupp$, donné par l'utilisateur, le problème de la recherche de régularités dans des transactions imbriquées consiste à rechercher tous les éléments E de niveau maximal tels que leur support est supérieur ou égal à $minSupp$.

Définie de cette manière, la problématique ressemble à celle de la recherche de règles d'association ou de motifs séquentiels. Cependant, contrairement à ces approches, les éléments manipulés sont composés de structures complexes. En effet, dans le cas des règles d'association, nous nous intéressons à la recherche

2. Nous prenons volontairement la notion de tâche, semblable aux tâches Ada, pour permettre d'illustrer la possibilité d'avoir deux tâches consécutives et dans un ordre bien précis.

de corrélation entre items, i.e. la recherche d'ensembles d'items maximaux dans la base. Dans le cas des motifs séquentiels, même si la notion de liste est prise en compte, il s'agit de liste d'ensembles d'items où les items sont atomiques.

Exemple 12 *Pour illustrer la composante de recherche dans des transactions imbriquées, considérons une base de données constituée d'un ensemble de transactions contenant des déclenchements de méthodes (par exemple, une base similaire pourrait être obtenue en analysant les fichiers log de règles actives).*

Id	*Transactions*
t_1	*<create(client), modify(order.status), <modify (client.credit), modify (client.trusted)>, modify(amount), modify(order.status), <modify (client.credit), modify (client.trusted)> >*
t_2	*<create(client), modify(amount), <modify(order.status), <modify(client.credit), modify(client.trusted)>, modify(client.credit)>, modify(status), <modify(client.credit), modify(client.trusted)>, modify(amount)>*
t_3	*<create(client), <modify (client.credit), modify (client.trusted)> >*
t_4	*<create(client), modify(order.status), <modify (client.credit), modify (client.trusted)> >*
t_5	*<create(client), modify(order.status), <modify (client.credit), modify (client.trusted)>, modify(amount),<modify (client.credit), modify (client.trusted)>, modify(order.amount) >*

Considérons que le support minimal spécifié par l'utilisateur est de 50%, c'est à dire que pour qu'une transaction imbriquée soit fréquente, il faut qu'elle apparaisse dans au moins trois transactions de la base. Nous trouvons les transactions imbriquées suivantes :

< create(client), modify(order.status), < modify(client.credit), modify(client. trusted)> modify(amount)> et < create(client), modify(client.credit), <modify(client.trusted) > >.

En effet, la première de ces transactions est imbriquée dans trois transactions de la base de données : t_1,t_2 et t_5. La seconde, par contre, est incluse dans toutes les transactions de la base. □

2.5.2 Un Système d'Extraction de Données Semi-Structurées sur le Web

Dans le cadre de la recherche de régularités dans des données semi-structurées issues du Web, le modèle de données utilisé est basé sur le modèle OEM que nous étendons pour pouvoir prendre en compte la notion de "liste-de".

La figure 2.8 décrit une partie des informations sur les restaurants dans la Bay Area [Abi97]. Chaque cercle avec le texte associé représente un objet et son identificateur. Les arcs et les étiquettes associés représentent des références d'objet. Par exemple, *value(&44) = "El Camino Real"* et *value(&19)="setof"*, spécifie que *category*, *name* et *address* peuvent ne pas être ordonnés.

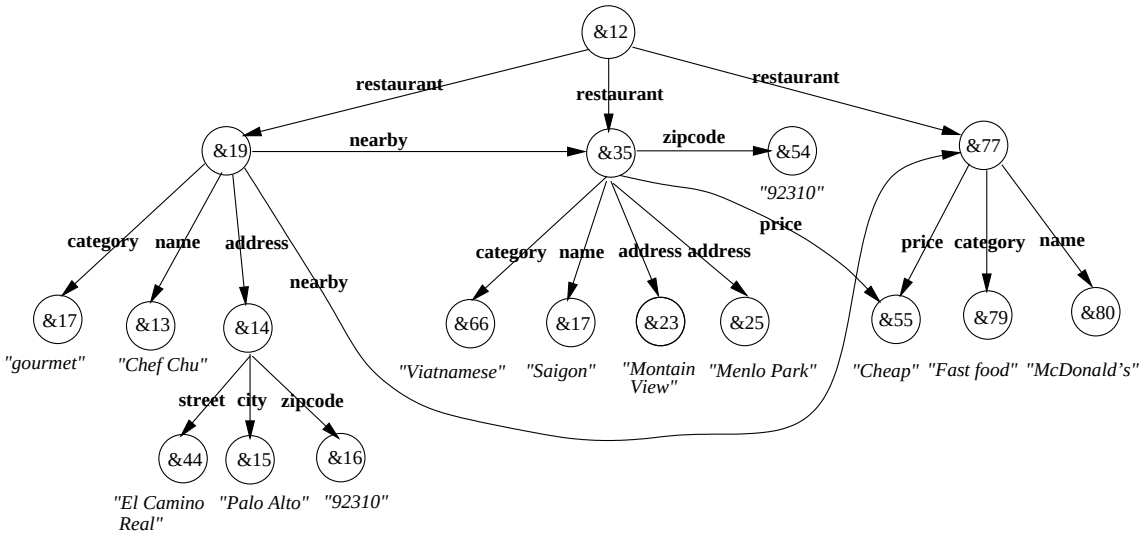

FIG. 2.8 – *Un exemple de graphe OEM*

Un chemin simple dans le graphe est une séquence alternative d'objets et d'étiquettes :

$$< o_1 l_1 o_2 ... o_{k-1} l_{k-1} o_k >$$

commençant et se terminant avec des objets et dans laquelle chaque étiquette a pour nœud de départ et d'arrivée respectivement le nœud qui le précède immédiatement et le nœud qui lui succède immédiatement. Le nombre d'étiquettes k parcouru depuis un nœud de départ jusqu'au dernier nœud dans le chemin correspond à la longueur du chemin. Un chemin multiple ou *chemin* est un ensemble de chemins simples tels que l'objet source de ces chemins soit le même.

Soit DB un ensemble de transactions où chaque transaction T est composée d'un identifiant et d'un chemin inclus dans le graphe OEM. Toutes les transactions sont triées par ordre croissant. La figure 2.9 illustre un exemple d'une base de données de transactions contenant des expressions de chemins.
Soit *supp(p)*, une valeur de support pour un chemin correspondant au nombre d'occurrences de ce chemin dans la base de données DB. En d'autres termes, le support d'un chemin p est défini comme le pourcentage de toutes les transactions qui contiennent p. Une transaction contient p si est seulement si p est un sous-chemin de la transaction.

Etant donnée une valeur de support minimal spécifiée par l'utilisateur ($minSupp$), un chemin est *fréquent* si la condition $supp(p) \geq minSupp$ est vérifiée.
Comme nous nous intéressons à la recherche des plus longs chemins, nous précisons qu'un chemin p_m est un sous-chemin d'un chemin p_n si p_m est inclus dans p_n, où l'inclusion de chemin est définie de la manière suivante : un chemin $p_a = <o_{a_1} l_{a_1} o_{a_2} .. o_{a_{k-1}} l_{a_{k-1}} o_{a_n}>$ est inclus dans un chemin $p_b = <o_{b_1} l_{b_1} o_{b_2} .. o_{b_{k-1}} l_{b_{k-1}} o_{b_m}>$ si et seulement s'il existe des entiers $i_1 < i_2 < \ldots < i_n$ tels que $o_{a_1} = o_{b_{i_1}}$, $o_{a_2} = o_{b_{i_2}}, \ldots, o_{a_n} = o_{b_{i_n}}$.

Trans_id	chemins
t_1	$person : \{identity: \{name: \perp, address; \perp\}\}$
t_2	$person : \{identity: \{name: \perp, address: < street: \perp, zipcode: \perp >,$ $company: \perp, director: < name: \perp, firstname: \perp >\}\}$
t_3	$person : \{identity: \{id: \perp, address: < street : \perp, zipcode : \perp >\}\}$
t_4	$person : \{identity: \{name: \perp, address: \perp, company: \perp\}\}$
t_5	$person : \{identity: \{name: \perp, address; \perp\}\}$
t_6	$person : \{identity: \{name: \perp, address: < street: \perp, zipcode: \perp >,$ $director: < name: \perp, firstname: \perp >\}\}$

FIG. 2.9 – *Un exemple de base de données de transactions*

Etant donnée une base de données DB de transactions, le problème de la recherche de régularités dans des données semi-structurées consiste donc à rechercher tous les chemins maximaux qui sont dans DB et dont le support est supérieur à $minSupp$.

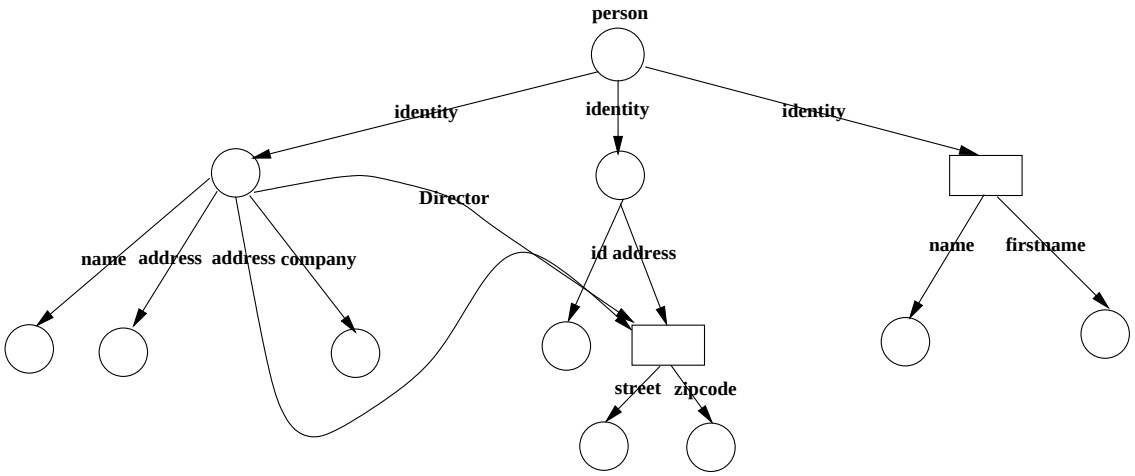

FIG. 2.10 – *Le graphe OEM associé*

Exemple 13 *Pour illustrer le problème de la recherche de régularités structurelles dans une base de données d'objets semi-structurés, considérons la base DB de la figure 2.9. Supposons que la valeur de support minimum spécifié par l'utilisateur soit 50%, c'est à dire que pour être fréquent un chemin doit apparaître dans au moins trois transactions. La figure 2.10 décrit le graphe OEM associé aux transactions de la base de données.*
Les seuls chemins fréquents dans DB sont les suivants :

identity: $\{name: \perp, address: \perp\}$,
et identity: $\{address: < street: \perp, zipcode: \perp >\}$.

Le premier car il est vérifié dans la transaction t_1 mais également dans les transactions t_4 et t_5. De la même manière

identity: $\{address: < street: \perp, zipcode: \perp >\}$

est inclus dans les transactions t_2, t_3 et t_6.
Par contre, le chemin

identity: $\{name: \perp, address: < street: \perp, zipcode: \perp >, director: < name: \perp, firstname: \perp >\}$

est vérifié par les transactions t_2 et t_6 mais n'est pas fréquent puisque le nombre de transactions qui possèdent ce chemin est inférieur à la contrainte de support minimal. □

2.6 Web Usage Mining

2.6.1 Un Système d'Extraction de Connaissances

Les principes généraux sont similaires à ceux du processus d'extraction de connaissances exposés dans [FPSSU96]. La démarche se décompose en trois phases principales. Tout d'abord, à partir d'un fichier de données brutes, un pré-traitement est nécessaire pour éliminer les informations inutiles. Dans la deuxième phase, à partir des données transformées, des algorithmes de data mining sont utilisés pour extraire les itemsets ou les séquences fréquents. Enfin, l'exploitation par l'utilisateur des résultats obtenus est facilitée par un outil de requête et de visualisation.
Les données brutes sont collectées dans des fichiers access log des serveurs Web. Une entrée dans le fichier access log est automatiquement ajoutée chaque fois qu'une requête pour une ressource atteint le serveur Web (*demon http*). Le fichiers access log peuvent varier selon les systèmes qui hébergent le serveur, mais présentent tous en commun trois champs : l'adresse du demandeur, l'URL demandée et la date à laquelle cette demande a eu lieu. Parmi ces différents types de fichiers, nous avons retenu dans cet article le format spécifié par le CERN et la NCSA [Con98], une entrée contient des enregistrements formés de 7 champs séparés par des espaces [NV96] :

```
host user authuser [date:time] ‘‘request’’ status bytes
```

La figure 2.11 illustre un extrait de fichier access log du serveur Web du Lirmm.

```
193.49.105.224 - - [29/Nov/2000:18:02:26 +0200] "GET /index.html HTTP/1.0 " 200 1159
193.49.105.224 - - [29/Nov/2000:18:02:27 +0200] "GET /PtitLirmm.gif HTTP/1. 0" 200 1137
193.49.105.224 - - [29/Nov/2000:18:02:28 +0200] "GET /acceuil_fr.html HTTP/1. 0" 200 1150
193.49.105.224 - - [29/Nov/2000:18:02:30 +0200] "GET /venir/venir.html HTTP/1.0" 200 1141
193.49.105.225 - - [29/Oct/2000:18:03:07 +0200] "GET /index.html HTTP/1.0" 200 1051
193.49.105.225 - - [16/Oct/2000:20:34:32 +0100] "GET /form/formation.html HTTP/1.0" 200 14617
193.49.105.225 - - [31/Oct/2000:01:17:40 +0200] "GET /form/formation.html#doct HTTP/1.0" 304 -
193.49.105.225 - - [31/Oct/2000:01:17:42 +0200] "GET /form/theses2000.html HTTP/1.0" 304 -
193.49.105.56 - - [22/Nov/2000:11:06:11 +0200] "GET /lirmm/bili/ HTTP/1.0" 200 4280
193.49.105.56 - - [22/Nov/2000:11:06:12 +0200] "GET /lirmm/bili/rev_fr.html HTTP/1.0" 200 2002
193.49.105.56 - - [07/Dec/2000:11:44:15 +0200] "GET /ress/ressources.html HTTP/1.0" 200 5003
```

FIG. 2.11 – *Exemple de fichier access log*

Deux types de traitements sont effectués sur les entrées du serveur log. Tout d'abord, le fichier access log est trié par adresse et par transaction. Ensuite une étape d'élimination des données "non-intéressantes" est réalisée. Au cours de la phase de tri et afin de rendre plus efficace le traitement de l'extraction de données, les URL et les clients sont codés sous forme d'entiers. Toutes les dates sont également traduites en temps relatif par rapport à la plus petite date du fichier.

Définition 6 Soit *Log* un ensemble d'entrées dans le fichier access log. Une entrée g, $g \in Log$, est un tuple $g =< ip_g,\{(l_1^g.URL,l_1^g.time),...,(l_m^g.URL,l_m^g.time)\} >$ tel que pour $1 \leq k \leq m$, $l_k^g.URL$ représente l'objet demandé par le client g à la date $l_k^g.time$, et pour tout $1 \leq j < k$, $l_k^g.time > l_j^g.time$.

La figure 2.12 illustre un exemple de fichier obtenu après la phase de pré-traitement. A chaque client correspond une suite de "dates" (événements) et la traduction de l'URL demandée par ce client à cette date.

Client	d1	d2	d3	d4	d5
1	10	30	40	20	30
2	10	30	60	20	50
3	10	70	30	20	30

FIG. 2.12 – *Exemple de fichier résultat issu de la phase de pré-traitement*

L'objectif est alors de déterminer, grâce à une phase d'extraction, les séquences de ce jeu de données, qui peuvent être considérées comme fréquentes selon la définition 4 (page 52). Les résultats obtenus sont du type `< ( 10 ) ( 30 ) ( 20 ) ( 30 ) >` (ici avec un support minimum de 100% et en appliquant les algorithmes de fouille de données sur le fichier représenté par la figure 2.12). Ce dernier résultat, une fois re-traduit en termes d'URL, confirme la découverte d'un comportement commun à *minSup* utilisateurs et fournit l'enchaînement des pages qui constituent ce comportement fréquent.

Dans notre contexte, en analysant les informations stockées sur le serveur Web, un exemple de règle d'association peut être : *50 % des clients qui ont visité les URL* `plaquette/info-f.html` *et* `labo/infos.html` *ont également visité* `situation.html` ou bien *85% des clients qui ont visité les URL* `iut/general.html/departement` *et* `info/program.html` *ont également consulté l'URL* `info/debouches.html`.

L'extraction de motifs séquentiels permet, d'autre part, de mettre en évidence le type de relation suivant : *60 % des clients qui ont visité* `/jdk1.1.6/docs/api/Package-java.io.html` *et* `/jdk1.1.6/docs/api/java.io.BufferedWriter.html`, *ont également visité, dans les 30 jours suivants, l'URL* `/jdk1.1.6/docs/relnotes/deprecatedlist.html` ou *34 % des clients ont visité* `/relnotes/deprecatedlist.html` *entre le 20 septembre et le 30 novembre.*

2.6.2 Modification Dynamique de Site Web

Les règles et motifs découverts constituent une connaissance très adaptée à l'optimisation dynamique de l'organisation hypertexte d'un serveur. Très utiles dans des applications telles que la conception

de catalogue "*on-line*" pour le commerce électronique, ces techniques exploitent les informations pertinentes sur le comportement des utilisateurs pour réorganiser un site afin d'améliorer son efficacité. Par exemple, les informations proposées peuvent être utilisées par un serveur Proxy dans le but d'améliorer l'accès aux pages. Quand une requête atteint le serveur Web, la requête est envoyée au Proxy qui charge les documents désirés. Les motifs séquentiels découverts peuvent alors être utilisés pour pré-charger les documents pour les utilisateurs [CKL97].

2.7 Web Usage Mining Inter-Sites

Une des caractéristiques incontournables de la structure du Web à l'heure actuelle, réside dans les possibilités qu'offrent les sites de passer de l'un à l'autre quand leurs centres d'intérêts le justifient. Ce type de liens, qui représente ce que l'on peut qualifier de partenariat entre sites web, apporte un nouvel élément qui se traduit au niveau des fichiers access log : le nombre d'utilisateurs communs aux sites qui pratiquent le partenariat est croissant. De la même façon que le cumul des données a créé le besoin de méthodes pour la fouille de données, ce phénomène de partenariat en en train de créer le besoin de méthodes pour le Web Usage Mining Inter-Sites.

2.7.1 Mise en place de la problématique : identification des enjeux

L'objectif du Web Usage Mining Intersites consiste à extraire des motifs séquentiels qui prennent place sur deux sites simultanément. Ces deux sites, A et B, ont tous deux mis en place des liens de partenariat qui permettent à l'utilisateur d'être redirigé, par exemple, du site A vers le le site B lorsque cela avantage les deux sites partenaires. Si l'on considère les clients que les deux sites ont en commun, et que l'on filtre les fichiers log de A et B pour ne retenir que ces clients, on obtient alors un fichier log du site virtuel $A \cup B$. C'est l'analyse de ce fichier et ses conséquences qui motivent la définition de cette problématique car les découvertes peuvent alors présenter deux avantages majeurs. Le premier de ces avantages est une meilleure qualité des motifs retenus, alors que le second réside dans le caractère immédiat de l'impact des résultat (en matière de modification dynamique de site).

Une qualité de connaissance accrue

Après la mise en place du site virtuel $A \cup B$, la séquence de navigation d'un utilisateur $C1$ commun aux deux sites est de la forme : <$s1$=`Navigation sur` A, $s2$=`Navigation sur` B>. Il s'agit en fait de la concaténation de deux séquences de navigation $s1$ et $s2$, enregistrées sur les fichiers access log de A et B. De plus, $s2$ correspond à la suite directe des opérations faites par $C1$ sur B, après avoir navigué selon le schéma exprimé par $s1$ sur A (i.e. $s2$ représente les opérations faites par $C1$ sur B, après que le dernier objet de $s1$ ait redirigé $C1$ vers le premier objet de $s2$).

Lorsque l'on filtre les données inscrites dans le fichier access log de B, afin de ne garder que celles qui sont issues de navigations qui suivent le partenariat entre A et B, le fichier considéré ne concerne qu'une seule classe de la population totale de B. A partir de cette classification directe, les résultats que fournit un processus de fouille de données sur ce sous-ensemble de la population, contiennent des informations présentant une confiance plus élevée, mais vont également révéler des fréquents plus

riches. L'exemple 14 donne une illustration de ce phénomène.

Exemple 14 *Considérons les fichiers access log des sites $SF1$, $SF2$ et SV illustrés par la figure 2.13. Un processus d'extraction de motifs séquentiels fréquents, avec un support de 100%, appliqué sur le fichier log de SV, donne le motif séquentiel suivant :* `<(i) (m) (w)>`*. Considérons à présent le fichier log de SV filtré par la sélection suivante : ne garder que les visiteurs ayant suivi un lien de partenariat entre $SF1$ et SV (i.e. dont la séquence de navigation commmence par* `(From_SF1)`*. On obtient alors un nouveau fréquent* `<(From_SF1) (i) (l) (m) (w)>`*. Ce motif séquentiel fréquent, propose une qualité de connaissance supérieure à celle obtenue dans le cas général, car il est dédié aux seuls visiteurs envoyés par le site $SF1$.*

Log $SF1$

Client	URL	Date
$C1$	a	01/01/2001 15h50:25s
$C1$	b	01/01/2001 15h50:28s
$C1$	c	01/01/2001 15h51:10s
$C1$	d	01/01/2001 16h00:55s
$C1$	To_SV	01/01/2001 16h01:10s
$C2$	a	03/01/2001 17h50:25s
$C2$	c	03/01/2001 17h50:28s
$C2$	d	03/01/2001 17h51:10s
$C2$	e	03/01/2001 18h00:55s
$C2$	To_SV	03/01/2001 18h01:10s
$C4$	u	04/01/2001 14h30:55s
$C4$	v	04/01/2001 14h31:10s

Log $SF2$

Client	URL	Date
$C3$	f	01/01/2001 15h50:28s
$C3$	g	01/01/2001 15h51:10s
$C3$	h	01/01/2001 16h00:55s
$C3$	To_SV	01/01/2001 16h01:10s

Log SV

Client	URL	Date
$C1$	From_SF1	01/01/2001 16h01:11s
$C1$	i	01/01/2001 16h01:50s
$C1$	j	01/01/2001 16h11:11s
$C1$	l	01/01/2001 16h11:50s
$C1$	m	01/01/2001 16h12:40s
$C1$	w	01/01/2001 16h12:55s
$C2$	From_SF1	03/01/2001 18h01:11s
$C2$	i	03/01/2001 18h01:50s
$C2$	k	03/01/2001 18h02:11s
$C2$	l	03/01/2001 18h02:50s
$C2$	m	03/01/2001 18h02:53s
$C2$	w	03/01/2001 18h03:10s
$C3$	From_SF2	01/01/2001 16h01:11s
$C3$	i	01/01/2001 16h01:50s
$C3$	z	01/01/2001 16h11:11s
$C3$	m	01/01/2001 16h11:50s
$C3$	w	01/01/2001 16h12:40s
$C3$	k	01/01/2001 16h12:55s

FIG. 2.13 – *access log des sites $SF1$, $SF2$ et SV*

Les chances de cibler le comportement d'un utilisateur sont alors augmentées dans la mesure où, pour prédire les comportements possibles de cet utilisateur, nous considérons des fréquents calculés uniquement en fonction de ses prédécesseurs, et non pas les comportements fréquents de tous les utilisateurs. L'utilisation des fréquents ainsi obtenus, sera donc plus efficace grâce à son adéquation avec la catégorie à laquelle appartient le client observé. Les liens proposés par le système de modification dynamique des liens hypertexte, ne seront pas ceux que l'on propose à tout utilisateur en fonction de tous les autres, mais ceux que l'on propose à un utilisateur de la catégorie α, en fonction du comportement fréquent

β des utilisateurs de cette même catégorie.

Toutefois, si l'on veut exploiter ces résultats au cours de la navigation d'un client C, il faut procéder à au moins trois étapes (en continuant l'exemple 14) :

1. Accepter la première demande de C, à savoir $From\_F1$. Cette page étant la même pour tous les utilisateurs de sa catégorie, on ne peut rien en déduire.
2. Accepter la seconde page $p2$, demandée par C. Une fois cette étape terminée, on peut commencer à prédire le comportement de C.
3. Accepter la troisième page $p3$, demandée par C et en fonction de sa demande précédente et ajouter sur $p3$ les liens vers les pages les plus demandées par ces prédecesseurs.

L'utilisation du Web Usage Mining Inter-Sites peut pourtant répondre à des impératifs plus exigeants que l'adéquation entre la catégorie à laquelle appartient l'utilisateur, et les liens qui lui sont proposés. Le fichier $Log+$ obtenu à partir des fichiers log de $SF1$ et SV, permet de tenir compte de l'historique de l'utilisateur observé, y compris sa navigation sur le site l'ayant envoyé sur SV. C'est en tenant compte de cet historique que l'on peut prétendre fournir des liens dès l'arrivée du client sur le site SV, accordant ainsi un impact immédiat aux résultat obtenus après l'analyse du fichier log de $A \cup B$.

L'impact immédiat des résultats

Tenir compte de l'historique d'un client sur le site A lorsqu'il demande une page sur le site B permet des tentatives de prédictions dès son arrivée sur le site B. En effet, cet historique peut soit correspondre à un des comportements fréquents observés chez ses prédecesseurs, soit rester neutre. Dans le cas d'une correspondance, le site B peut non seulement cibler le comportement du client en fonction de la catégorie à laquelle il appartient (cf paragraphe çi-dessus), mais surtout faire des propositions de pages à consulter pour cet utilisateur (ou modifier le site de la manière qui conviendra le mieux) en un nombre d'étapes minimum. Ce paragraphe, présenté sous forme d'exemple, illustre les possibilités offertes par le Web Usage Mining Inter-Sites sur le plan des modifications dynamiques de liens hypertextes.

Reprenons les fichiers log illustrés par la figure 2.13. Les séquences de navigation des utilisateurs sur ces sites sont alors :

$C1$: `<(a) (b) (c) (d) (To_SV)>` sur $SF1$ et `<(From_SF1) (i) (j) (l) (m) (w)>` sur SV
$C2$: `<(a) (c) (d) (e) (To_SV)>` sur $SF1$ et `<(From_SF1) (i) (k) (l) (m) (w)>` sur SV
$C3$: `<(f) (g) (h) (To_SV)>` sur $SF2$ et `<(From_SF2) (i) (z) (m) (w) (k)>` sur SV

Si l'on s'intéresse au partenariat existant entre les sites $SF1$ et SV, le fichier $Log+$ correspondant à ces deux sites et sur lequel nous allons procéder à une phase d'extraction de connaissances, est illustré de manière synthétique (i.e. sans les dates mais en conservant l'ordre d'apparition des objets demandés) par la figure 2.14.

Un processus de fouille de données, executé avec un support $minSup$=100% sur ce fichier, permet d'obtenir le comportement fréquent suivant : `<(a) (c) (d) (To_SV) (From_SF1) (i) (l) (m) (w)>`. Les trois étapes nécessaires, décrites dans le paragraphe précedent, à la prise en compte des résultats avant d'exploiter les motifs séquentiels obtenus chez un client C dont le comportement s'apparente au comportement fréquent de ses prédécesseurs, sont alors réduites à une seule étape : Accepter la demande

Client	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11
$C1$	a	b	c	d	To_SV	From_SF1	i	j	l	m	w
$C2$	a	c	d	e	To_SV	From_SF1	i	k	l	m	w

FIG. 2.14 – *Fichier Log+, correspondant aux fichiers $LogSF1$ et $LogSV$ de la figure 2.13*

faite par C de la page $From\_SF1$ et renvoyer cette page modifiée (i.e. lui ajouter des liens vers les pages (i) et/ou (l) et/ou (m) et/ou (w)).
D'un point de vue global, les liens ajoutés à la première page demandée par C, seront dépendants du nombre et des types d'historiques fréquents que présentent les clients qui passent de $SF1$ a SV comme l'illustre l'exemple 15.

Exemple 15 *Considérons que deux fréquents sont issus de l'analyse du fichier Log+ entre $SF1$ et SV :*
`<(a) (b) (d) (To_SV) (From_SF1) (e) (g)>` *et* `<(a) (c) (d) (To_SV) (From_SF1) (f) (h)>`.
Si un client passe de $SF1$ à SV (toujours en restant dans le cadre du partenariat) et présente à son arrivée sur SV un historique qui correspond au premier fréquent, alors la première page qu'il consulte sur SV ($From\_SF1$) peut être augmentée par des liens vers (e) et/ou (g). Son comportement sur $SF1$ a permis de le distinguer des utilisateurs qui ont eu un comportement correspondant au second fréquent (ou à un comportement neutre).

Ensuite le comportement de C sur SV permettra de prévoir son comportement à venir avec une précision supplémentaire par rapport au Web Usage Mining Intra-Site puisque l'historique sera encore pris en compte à chaque tentative de faire correspondre son comportement à celui de ses prédécesseurs.

2.7.2 Le partenariat

Reprenons l'exemple de partenariat décrit ci-dessus. L'exemple 16 illustre la façon dont les fichiers access logs des sites Web mis en jeu peuvent être fusionnés.

Exemple 16 *La figure 2.15 illustre la façon dont les clients sont sélectionnés, à partir des fichiers access log de $SF1$ et SV, dans le but de construire le fichier access log qui leur est commun. Ce nouveau fichier "Log+" contient les entrées correspondant aux visiteurs de $SF1$ qui ont utilisé un lien de $SF1$ vers SV.*

Cette nécessité de mettre en commun les fichiers access log, nous conduit dans un premier temps à préciser la notion de *log* d'abord introduite par la définition 6 :

Définition 7 Soit *Log* un ensemble d'entrées dans le fichier access log. Une entrée g, $g \in Log$, est un tuple $g =< ip_g.s,\{(l_1^g.URL,l_1^g.time),...,(l_m^g.URL,l_m^g.time)\} >$ tel que pour $1 \leq k \leq m$, $l_k^g.URL$ représente l'objet demandé par le client g à la date $l_k^g.time$, et pour tout $1 \leq j \leq m/\ k > j$, $l_k^g.time > l_j^g.time$. s, qui représente le numéro de session de la séquence de navigation de g, est déterminé par l'algorithme 2.1.

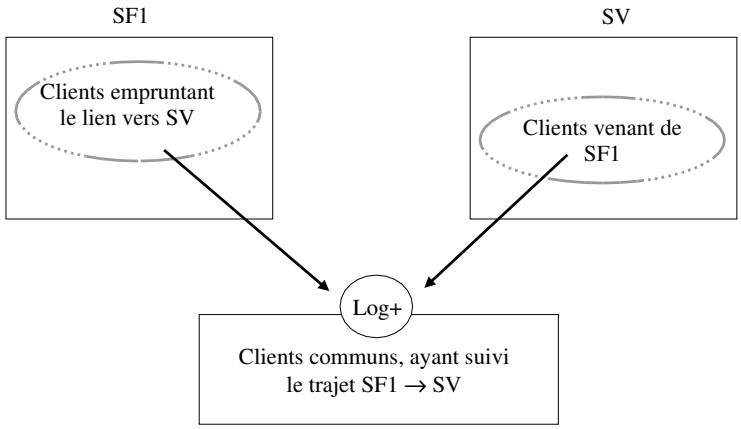

FIG. 2.15 – *Un exemple de mise en commun des fichiers access log*

Site	URL	Date
$SF1$	a	01/01/2001 15h30:25s
$SF1$	b	01/01/2001 15h30:28s
$SF1$	To_SV	01/01/2001 15h31:10s
SV	From_SF1	01/01/2001 15h31:11s
SV	d	01/01/2001 15h31:50s
$SF1$	a	03/01/2001 16h00:25s
$SF1$	c	03/01/2001 16h00:28s
$SF1$	To_SV	03/01/2001 16h01:10s
SV	From_SF1	03/01/2001 16h01:11s
SV	e	03/01/2001 16h01:50s

FIG. 2.16 – *Historique du client $C1$, sans le concept de session*

Dans cette définition, la différence réside dans l'ajout d'un numéro de session s, associé au client. Ce numéro de session permet de décliner le client en plusieurs versions, en fonction du nombre de ses visites sur le site. Il est déterminé par un délai maximum, fixé par l'utilisateur, qui permet de décider si un client est encore dans un même objectif de navigation, ou si il s'est reconnecté sur le site après un temps N. Ce temps fixé par l'utilisateur, si il est dépassé entre deux actions du client (par exemple deux actions séparées par un intervalle de 2 jours), permet de conclure que ce client peut être considéré comme une nouvelle entrée dans le fichier access log, afin d'éviter des confusions comme celle décrite dans l'exemple 17.

Exemple 17 *La figure 2.16 illustre un cas de figure sans la notion de session. On peut y constater que le client $C1$ s'est connecté sur le site $SF1$ à deux jours d'intervalle et en suivant le lien vers SV systématiquement. La séquence de navigation de $C1$, d'après les fichiers logs illustrés par la figure 2.17, devient alors* `<(a) (b) (To_SV) (a) (c) (To_SV) >`*, sur $SF1$ et* `<(From_SF1) (d) (From_SF1) (e)>`

Log $SF1$

Client	URL	Date
$C1$	a	01/01/2001 15h30:25s
$C1$	b	01/01/2001 15h30:28s
$C1$	To_SV	01/01/2001 15h31:10s
$C1$	a	03/01/2001 16h00:25s
$C1$	c	03/01/2001 16h00:28s
$C1$	To_SV	03/01/2001 16h01:10s

Log SV

Client	URL	Date
$C1$	From_SF1	01/01/2001 15h31:11s
$C1$	d	01/01/2001 15h31:50s
$C1$	From_SF1	03/01/2001 16h01:11s
$C1$	e	03/01/2001 16h01:50s

FIG. 2.17 – *access log des sites $SF1$ et SV, sans concept de session*

Log $SF1$

Client	URL	Date
$C1.1$	a	01/01/2001 15h30:25s
$C1.1$	b	01/01/2001 15h30:28s
$C1.1$	To_SV	01/01/2001 15h31:10s
$C1.2$	a	03/01/2001 16h00:25s
$C1.2$	c	03/01/2001 16h00:28s
$C1.2$	To_SV	03/01/2001 16h01:10s

Log SV

Client	URL	Date
$C1.1$	From_SF1	01/01/2001 15h31:11s
$C1.1$	d	01/01/2001 15h31:50s
$C1.2$	From_SF1	03/01/2001 16h01:11s
$C1.2$	e	03/01/2001 16h01:50s

FIG. 2.18 – *access log des sites $SF1$ et SV, avec concept de session*

sur SV. Ces séquences présentent sans doute un intérêt pour les sites observés indépendamment, mais leur niveau de détail est insuffisant pour garder une trace historisée des passages de $C1$ entre $SF1$ et SV. Grâce à la notion de session décrite par la définition 7, les entrées dans le fichier log de $SF1$ distinguent les deux types de navigation de $C1$, en déclinant deux versions de cet utilisateur (i.e. $C1.1$ et $C1.2$) qui seront alors considérées comme deux clients différents. Cette version, exploitable pour le Web Usage Mining Inter-Sites, des fichiers access log est illustrée par la figure 2.18, et correspond au même historique qui, sans ce concept, serait problématique. Le client $C1$ se voit donc décliné en deux versions ($C1.1$ et $C1.2$) et les séquences de navigation de ces nouveaux clients sur $SF1$ et SV sont alors :

- *pour $C1.1$:* `<(a) (b) (To_SV)>` *sur $SF1$ et* `<(From_SF1) (d)>` *sur SV.*
- *pour $C1.2$:* `<(a) (c) (To_SV)>` *sur $SF1$ et* `<(From_SF1) (e)>` *sur SV.*

Pour terminer la présentation de cette notion de session, précisons que le fichier access log sans sessions de SFx, peut être retrouvé sans difficulté en supprimant le $.s$ dans le fichier. Ainsi $C1.1$ et $C1.2$ (pour reprendre l'access log de $SF1$ dans l'exemple 17) deviennent à nouveau $C1$.

La définition 8, présente le concept de $Log+$, qui regroupe pour deux sites, les visiteurs qu'ils ont en commun (i.e. le fichier "$Log+$" de la figure 2.15).

Définition 8 Soit Log_A et Log_B l'ensemble des entrées des fichier log des sites A et B. Soit $Log+$ le fichier log crée à partir de Log_A et Log_B.
Une entrée g, $g \in Log+$, est un tuple $g =< ip_g.s,\{(l_1^g.URL,l_1^g.time),...,(l_m^g.URL,l_m^g.time)\} >$ respectant les propriétés suivantes :

- $\exists\ i,\ 1 \leq i \leq m /\ l_i^g.URL$=To_B. Si $i < m$ alors $(l_{i+1}^g.URL)$=From_A
  et $\forall z > i+1 /\ (l_z^g.URL) \neq$From_A.
- $\forall g_A \in Log_A$ si $l_m^{g_A}.URL =$To_B, avec m la longueur de g_A, alors $\exists g \in Log+ / \forall i \in (1..m)\ l_i^{g_A} = l_i^g$
- $\forall g_B \in Log_B$ si $l_1^{g_A}.URL =$From_A, avec m la longueur de g_B,
  alors $\exists g \in Log+$ de longueur n / $\forall i \in ((n-m)..n)\ l_i^{g_B} = l_i^g$

Le premier point de la définition 8 exprime le fait que toutes les entrées dans $Log+$ représentent la navigation d'un utilisateur qui est passé du site A à son site partenaire B en utilisant les liens du partenariat. Ce premier point montre aussi que si un échec est survenu pendant l'acheminement d'un client du site A vers le site B (pour cause de lien "cassé", problème réseau, sécurité, ou autre...) l'entrée est conservé dans le fichier $Log+$ car elle pourra permettre de déterminer l'importance, les causes et les solutions à envisager à ces failles dans les redirections. Enfin, il garantit que pour un client associé à une session, il ne peut exister, au plus, qu'une seule demande pour l'objet $From\_A$. Le second point assure que si un utilisateur du site A a emprunté un tel lien, alors il est le début d'une entrée dans le fichier $Log+$. Le troisième point assure que si un utilisateur du site B est arrivé grâce à un lien du partenariat, alors il termine une entrée de $Log+$. Finalement, le fichier $Log+$ peut être considéré comme le fichier access log du site virtuel constitué par $A \cap B$ restreint aux clients qui sont passés de A à B en suivant les liens du partenariat.

Exprimons les navigations des clients illustrées par la figure 2.18 sous forme de séquences de navigation. Le client $C1.1$, au cours de sa session, a réalisé le parcours suivant : `<(a) (b) (To_SV) (From_SF1) (d)>`. Le client $C1.2$, aura le parcours : `<(a) (c) (To_SV) (From_SF1) (e)>`. Ces deux séquences, associées aux clients qui les supportent et aux dates auxquelles les URLs sont demandées, forment les deux entrées du fichier $Log+$ correspondant aux fichiers log de $SF1$ et SV. Notons que sans le concept de session, ce fichier $Log+$ serait composé d'une seule entrée :
`<(a) (b) (To_SV) (a) (c) (To_SV) (From_SF1) (d) (From_SF1) (e)>` et que cette séquence ne nous permet pas, directement, de fournir des connaissances sûres, dans la mesure où les différents passages de $SF1$ à SV sont "brouillés".
Le problème du **Web Usage Mining Inter-Sites** consiste à découvrir les navigations fréquentes des utilisateurs communs à deux sites A et B en observant le site virtuel $A \cap B$, via le fichier $log+$.

2.8 Web Usage Mining Temps Réel

Il existe des points de vue sur le Web Usage Mining Temps Réel qui reposent sur des applications successives d'un algorithme de fouille de données incrémentale ([]). Pour mettre en œuvre ce type de

function *accepterRequête*
Input : Un client $Cn.x$, dans sa session x, demandant un objet sur le site A ou B.
N la durée spécifiée par l'utilisateur pour la durée d'une session.
Output : $Cn.y$ le client avec un éventuel nouveau numéro de session.
if *Cn demande From_ A sur B* **then**
// Cn passe du site A au site partenaire B par un lien de partenariat
$y = x$; // On ne change rien, Cn garde son numéro de session;
endif
return($Cn.y$) **if** *dernierObjet(Cn)* $\in$ B **and** *Cn demande une page sur A* **then**
// Cn passe du site B au site partenaire A, c'est une nouvelle session
$y = x + 1$; // On incrémente le numéro de session
endif
return($Cn.y$) **if** *dateDernierObjet(Cn)*$>N$ **and** *Cn demande une page sur A* **then**
// Cn reste sur A mais sa dernière action a une date plus grande que N : nouvelle session
$y = x + 1$; // On incrémente le numéro de session
endif
return($Cn.y$) **end function** *accepterRequête*

Algorithme 2.1: *Mise à jour des numéros de session*

solution, il faut réaliser que la notion de temps réel reste limitée par le temps d'exécution d'une composante "fouille de données incrémentale" car l'algorithme ne pourra être re-employé que lorsque son temps d'exécution le permettra. Les défauts de cette approche prennent alors une amplitude considérable si l'on considère la taille du fichier access log à analyser, qui fera, en augmentant, disparaître de façon proportionnelle l'aspect temps réel recherché.

Le but de cette page n'est pas de faire une présentation des techniques connues en matière de métaheuristiques, mais plutôt d'envisager une technique de ce type pour la fouille de données et d'en identifier les failles.

L'extraction de motifs séquentiels peut faire appel à deux types de solutions :

1. Les algorithmes qui fournissent une solution complète, après avoir exploré l'espace des solutions possibles de la façon la plus pertinente possible.
2. Les heuristiques, qui garantissent un résultat fiable, approchant l'optimum global et affichant des temps d'exécution largement inférieurs en comparaison des algorithmes du premier point.

Les solutions du premier type sont nombreuses et basées sur des techniques comme la méthode générer-élaguer (cf figure 2.19), ou une exploration en mémoire de la base de données ([HPY00], les arbres suffixés). Malheureusement, si la taille du problème augmente, de telle solutions peuvent être mises en échec et ne fourniront "jamais" le résultat (l'exploration des possibilités devient trop coûteuse).

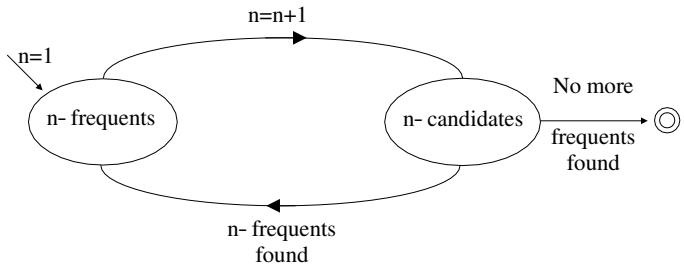

FIG. 2.19 – *La méthode Générer-Élaguer*

Certaines méthodes proposent alors de fournir une solution très approchante (voire égale dans la plupart de cas) de l'optimum global [Bay98, Toi96, GMS97]. Ces méthodes correspondent au deuxième type de solutions et ont pour but de répondre aux deux cas de difficultés majeures que l'on peut rencontrer :

1. **La taille du résultat** interdit toute enumération des sous parties du résultat (ce que fait la méthode générer-élaguer).
2. **La quantitié de données** (taille de la base) à explorer est bien trop grand pour employer une méthode de type "exploration en mémoire".

Il existe cependant un troisième cas, qui lui aussi empèche l'utilisation des méthodes fournissant une solution optimale et qui reste fortement lié à la problématique du Web Usage Mining : **une évolution rapide des données à traiter**.

En effet, le site étudié peut voir le comportement de ses clients évoluer rapidement alors que son fichier access log est encore en cours d'analyse. un algorithme de type incrémental (même appliqué aussi fréquement que son temps d'exécution le permet) ne peut prétendre être une solution temps réel, ce qui nous conduit à mettre au point un algorithme stochastique d'optimisation combinatoire pour résoudre le problème de l'extraction de motifs séquentiels dans le cadre du Web Usage Mining.

La figure 2.20 illustre une méthode possible pour résoudre le problème de l'extraction de motifs séquentiels dans les bases de données en utilisant une méta-heuristique. En considérant DB (étape 4) comme notre access log, il est effectivement possible d'employer une telle méthode pour découvrir les comportements fréquents des utilisateurs d'un site web. Le point de départ serait une population de fréquents réduite aux items (URL) fréquents (les plus demandées) (étape 1). Ensuite des opérateurs de voisinage proposent un ensemble de solutions candidates (étapes 2 et 3) et leur validation sur la base de données (étape 4) permet de découvrir une solution plus fine (étapes 5 et 6). Si l'on applique à nouveau les opérateurs de voisinage sur ces fréquents, on peut alors affiner encore les résultats et ce processus se répète jusqu'à stabilité de la solution (ensemble de fréquents) proposée.

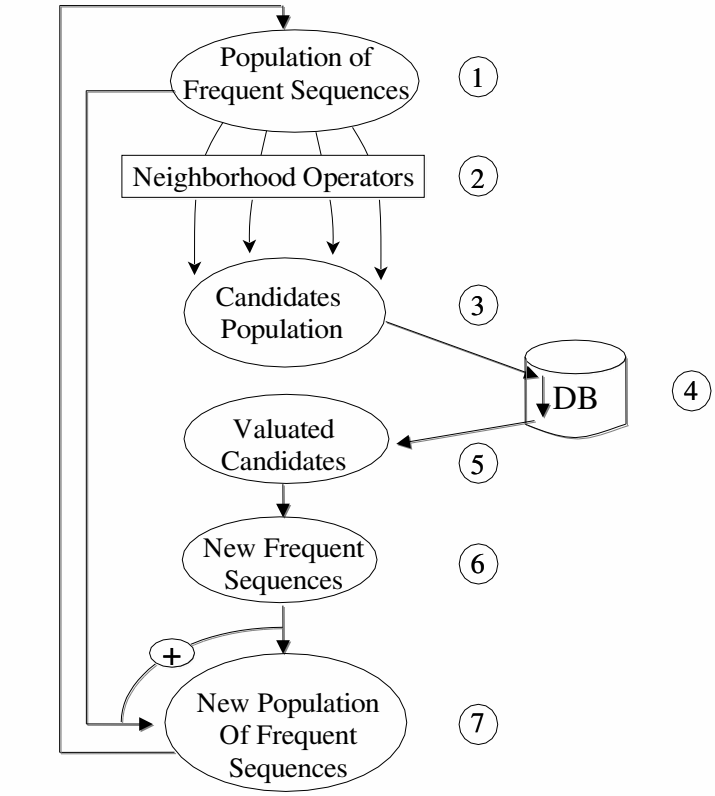

FIG. 2.20 – *Proposition de méta-heuristique pour la fouille de données*

Cette solution présente pourtant un défaut majeur : le nombre de passes nécessaires sur la base de données pour évaluer chaque solution proposée est bien trop grand. En effet une méthode de type générer-élaguer ne demande que k passes, avec k=taille du fréquent le plus grand avant de fournir les résultats. Une méta-heuristique demande plusieurs dizaines (centaines) de validations sur la base de données (c'est la fonction objective, pilier des algorithmes stochastiques qui est censée s'exécuter en un temps minimum).

Pour qu'une telle solution soit envisageable, il faudrait que la puissance de calcul disponible (pour comparer chaque candidat avec chaque séquence de la base) augmente en même temps que la taille de la base. C'est ce que propose la méthode exposée dans le chapitre 10, et qui fonctionne actuellement à l'adresse. `http://www.kdd-tools.com`.

Chapitre 3

Travaux antérieurs

Les problèmes de recherche de motifs, de sous-mots ou encore les problèmes de recherche de sous-mots communs sont à la base de nombreux travaux de recherche. Cet intérêt pour la détection de particularités dans un ensemble de données, trop riche à l'échelle des facultés humaines, révèle un besoin de plus en plus indispensable d'exploiter cette richesse. Ce besoin, amplifié par la masse de données enregistrée chaque jour, s'exprime par une volonté de filtrer, selon différents points de vue, la masse d'informations, afin de n'en garder que les valeurs significatives. Les critères principaux de ces explorations s'avèrent être les répétitions et les régularités (l'unicité pourrait aussi bien présenter un intérêt particulier). Synonymes de révélations, ces phénomènes attirent l'attention des méthodes d'IA qui, entre autres, cherchent à construire des raisonnements basés sur des données qu'un critère permet de mettre en valeur. La section 3.2.4 fait le point sur quelques unes de ces méthodes mais aussi sur l'apport que peuvent fournir les techniques de recherche de régularités pour des sciences autres que l'informatique. Les sections 3.1 et 3.2 gardent un axe plus précisemment orienté vers les problèmes liés à la fouille de données. Nous y présentons les travaux de recherche sur les règles d'associations, qui sont à l'origine des approches de recherche de motifs séquentiels. Plus particulièrement, la section 3.2 résume les travaux présentés par [AS95, SA96b] qui sont à la base des définitions et méthodes expliqués dans ce document, ainsi que les contributions qui s'articulent sur ce thème. Quelques travaux relatifs à la recherche incrémentale de motifs séquentiels sont décrits à la section 3.3.

Les travaux existants sur les applications que nous avons abordées sont décrits par les sections 3.4, pour le Schema Mining, et 3.5, pour le Web Usage Mining.

Enfin, une discussion est proposée par la section 3.6 pour souligner les qualités, comme les défauts, des travaux présentés dans cet état de l'art.

3.1 Règles d'associations

La découverte de règles à partir de données empiriques est, avant tout, une source d'activité majeure pour des domaines tels que l'IA et plus particulièrement les approches de machine learning. Le problème de la découverte de règles d'associations se distingue des objectifs recherchés par la plupart des taches en machine learning par plusieurs points qui rassemblent, entre autres, la quantité de données traitées ou leur nature (les processus de recherche de règles d'association ne font généralement pas cas des valeurs nulles ou des valeurs inutiles). Les règles d'associations ont un intérêt que des applications, telles que le marketing, trouvent flagrant. En effet, les bases de données actuelles peuvent stocker tant d'informations qu'elles engendrent des phénomènes de procédés marketing basés sur l'information massive (par opposition au marketing basé sur l'échantillonage mis en œuvre par les études de comportements observés ou encore les sondages). Les possibilités de stockage, en matière d'information, connaissent des progrès tels, qu'elles ont permis à toutes sortes d'organisations commerciales, l'enregistrement massif de données concernant les ventes, baptisées *basket data*.
Formellement, le problème des règles d'associations est présenté (cf. [AIS93]) de la façon suivante :

Soit $I = \{i_1, i_2, \ldots, i_m\}$ un ensemble d'items (au sens des définitions données dans la section 2.2.1. Soit D un ensemble de transactions, tel que chaque transaction T est un ensemble d'items vérifiant $T \subseteq D$.

T contient X, un ensemble d'items de I, si $X \subseteq T$. Une règle d'association est une implication de la forme $X \rightarrow Y$, où $X \subseteq I$, $Y \subseteq I$ et $X \cap Y = \emptyset$. La règle $X \rightarrow Y$ a un *support s* pour D si $s\%$ des transactions de D contiennent $X \cup Y$.

Cette définition est ensuite étendue par [AS94], qui permet à la partie *conséquence* d'une règle d'être composée de plusieurs items.

Exemple 18 *Un libraire peut découvrir que la règle (Star Wars)→(Empire Strikes Back, Return of The Jedi) est vérifié dans* 39% *(confiance) des achats effectués.* Cela signifie que $s\%$ (support) des achats contiennent les items "Star Wars", "Empire Strikes Back" et "Return of The Jedi" mais aussi que 39% des acheteurs de "Star Wars" ont aussi acheté "Empire Strikes Back" et "Return of The Jedi" ce qui peut être considéré comme une probabilité pour le futur.

Attirés par les résultats qu'elles promettent et les enjeux auxquels elles sont consacrées, un grand nombre d'algorithmes de fouille de données sont destinés à la recherche de règles d'association. Nous porterons, dans cette section, un regard couvrant l'ensemble des méthodes issues des travaux effectués, aux travers des différents courants qui ont émergé. Les différentes méthodes présentées ont toutes un intérêt que chaque application peut exploiter en fonction de ses caractéristiques.

3.1.1 Pionnières

Le premier algorithme destiné à résoudre le problème de la recherche de règles d'associations, selon la définition donnée dans cette section, est présenté dans [AIS93] et porte le nom de AIS (Agrawal, Imielinski, Swami). Dans cet algorithme, des candidats (un candidat est un itemset susceptible d'être fréquent dans la base de données) sont générés et testés à la volée, pendant une passe sur la base de donnée. Après avoir lu une transaction, l'algorithme détermine quels itemsets fréquents de la passe précédente sont contenus dans cette transaction. De nouveaux candidats sont alors générés par extension de ces itemsets fréquents avec des items de la transaction. Il s'agit là des premières pierres sur lesquelles va se construire la méthode générer-élaguer (que nous détaillons dans la section 3.2). Les candidats générés à partir d'une transaction sont ajoutés à l'ensemble des candidats construits pendant cette passe (ou leurs compteurs sont incrémentés s'ils appartiennent déjà à cet ensemble).
Dans le but, également, de construire des règles d'associations, mais motivés par les possibilités d'utiliser SQL pour calculer les itemsets fréquents, [HS93] propose l'algorithme SETM. Basé sur le principe d'AIS, cet algorithme utilise SQL pour calculer les itemsets candidats, grâce à une opération de jointure standard de SQL entre les fréquents. Cet approche à révélé des inconvénients dus à la taille excessive des candidats générés.
A partir de raisonnements plus fins concernant les inclusions de fréquents, [AS94] propose un nouvel algorithme qui reste encore une référence : Apriori. L'idée fondatrice de cet algorithme est alors la méthode générer-élaguer qui peut-être présentée, de manière informelle, comme ceci : puisque les fréquents ont entre eux des propriétés particulières, relatives à l'inclusion et puisque l'on sait qu'il existe une taille maximale de fréquents (le fréquent de plus grande taille dans la base), il faut procéder par étapes (passes sur la base de données). Pour cela, déterminer les fréquents de taille 1 (les items fréquents), construire les candidats de taille 2 (à partir des fréquents de taille 1), déterminer les fréquents de taille 2 par une phase qui détermine pour chaque transaction dans D, quels sont les 2-candidats

contenus dans cette transaction et réitérer (2-candidats $\rightarrow_{prune}$ 2-fréquents $\rightarrow_{gen}$ 3-candidats $\rightarrow_{prune}$ 3-fréquents $\rightarrow_{gen}$ 4-candidats ...) jusqu'à ce que l'on ne trouve plus aucun fréquent. De plus amples renseignements sur la méthode générer-élaguer sont donnés dans la section 3.2.

3.1.2 Minimisant le nombre de passes

Pour tenter d'accélérer la production des résultats, de nouvelles méthodes sont alors élaborées à partir de l'observation suivante : le nombre de passes sur la base de données est à l'origine d'un trop grand nombre d'accès disque , dont la lenteur n'est plus à discuter.
Pour envisager une solution à ce problème, la méthode Partition ([SON95]) propose de diviser la base de données afin de déterminer les fréquents en seulement deux passes. Pour cela l'algorithme détermine un nombre de partitions de tailles identiques et met en œuvre une méthode similaire à Apriori pour calculer en mémoire les fréquents locaux à chaque partition. Notons L^i l'ensemble des fréquents de la partition i. Du fait de la différence entre les données stockées dans chaque partition, les fréquents qu'elles contiennent sont également différents. Pour déterminer les fréquents de manière globale, une autre passe sur la base (*counting phase*) est nécessaire. Pour créer les candidats qu'il faut tester dans cette phase, l'ensemble des *candidats-globaux* $C^G = \bigcup_i L^i$ est construit comme l'union de tous les fréquents locaux. Les travaux de [GPW98] étendent l'algorithme de Partition, en ajoutant une structure de recherche plus efficace sur les ensembles fréquents.
[BMUT97] propose une solution destinée également à déterminer les k-fréquents en moins de k passes. Pour cela, l'algorithme DIC proposé, divise lui aussi la base en plusieurs parties de tailles égales. Supposons que la base soit divisée en trois parties et que les fréquents aient une taille maximale de trois items. DIC construit sur une première partie de la base tous les fréquents de longueur 1 rencontrés puis continue en créant les fréquents de longeur 2 à partir des 1-fréquents déjà extraits. Pendant qu'il construit les 2-fréquents, DIC continue de créer les 1-fréquents et commence à créer les 3-fréquents. Une fois la première passe terminée l'algorithme a obtenu les items fréquents, mais aussi les 2-fréquents aux environs des deux tiers de leur totalité et le premier tiers des 3-fréquents. La deuxième passe n'a pas besoin d'être achevée car le premier tiers de la base doit être vérifié pour les 2-fréquents (jusqu'ici seul les deux derniers tiers ont servis à les construire) et les deux premiers tiers suffisent à achever la construction des 3-fréquents (le troisième est déjà exploré pour les 3-fréquents). La figure 3.1 illustre le fonctionnement de cet algorithme.

Il existe également des méthodes stochastiques incomplètes qui peuvent être classées parmi les méthodes économes en accès disque. Citons par exemple [GMS97] qui propose une méthode stochastique basée sur une généralisation du concept d'hypergraphe transversal. [Toi96] utilise une méthode d'échantillonnage, Sampling, dont la première version se trouve dans [MTV94]. Cette méthode permet d'obtenir la plupart des fréquents (les auteurs affichent une très faible probabilité pour qu'un fréquent n'apparaisse pas).
Malgré les temps de réponse affichés pour ces algorithmes, il faut considérer les inconvénients engendrés par la compensation des accès disques par des calculs qui peuvent affecter leurs performances.

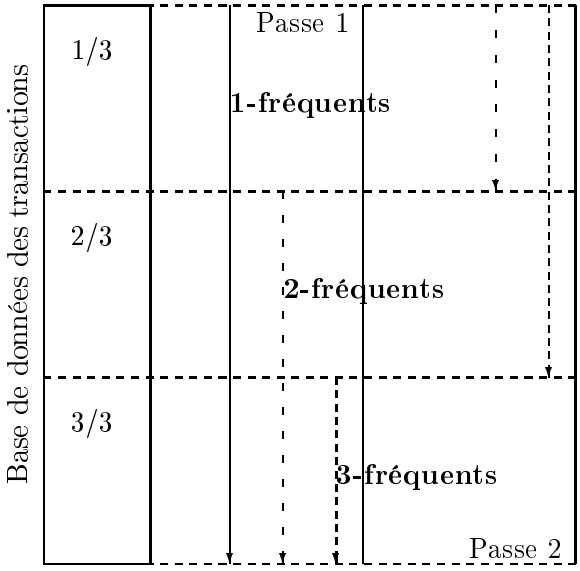

FIG. 3.1 – *1 passe + 2/3 passe avec l'algorithme DIC*

3.1.3 Adaptées à l'organisation des données

Les approches précédentes ont en commun le fait qu'aucune d'entre elles ne tient compte du fait que les données peuvent être corrélées. En effet dans [PBTL98], l'algorithme Close proposé offre des temps de réponse particulièrement performants sur des bases de données telles que les données de recensement. Si cet algorithme, basé sur l'utilisation de treillis de Galois, n'accélère pas les temps de réponse sur les benchmarks proposé par [AS94], il a l'avantage de garder les meilleures performances sur un type de données particulier : les données corrélées.

3.1.4 Rapides mais limitées aux bases de données de petite taille

[Wan97] propose une méthode d'extraction de règles d'association, basée sur l'utilisation de suffix tree. Si l'algorithme exposé semble afficher des temps de réponse performants, sa complexité en mémoire reste un point obscur. En effet, il est certain qu'une méthode basée sur des suffix tree nécessite un espace en $O(n)$ (la seule information disponible quand à la complexité en espace de ce travail est qu'il connaît une complexité en mémoire *linéaire*) avec n la taille de la base de données (cf. [WCM+94]). Il est cependant important de préciser que le travail mené par [Wan97] est la continuation de [WT96] qui exprimait la volonté de proposer une fouille de données incrémentale pour la recherche de règles d'associations. Ce but est, en effet, correctement atteint par la méthode proposée grâce à l'utilisation des suffix tree, bien adaptés à la problématique de l'incrémentalité.

Transaction	Items	Items fréquents (support décroissant)
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

FIG. 3.2 – *Base de données exemple pour la structure de FP-tree*

Sans génération de candidats

Les auteurs de [HPY00] considère que le goulot d'étranglement des processus d'extraction de type *A-priori*, se situe au niveau de la génération des candidats. Leur étude montre, en effet, que pour 10^4 items fréquents découverts, l'algorithme *A-priori* devra tester 10^7 candidats de taille 2 sur la base de données. De plus pour découvrir des séquences fréquentes de taille 100, cet algorithme pourra tester (et donc générer) jusqu'à 10^{30} candidats au total.

L'approche proposée par [HPY00] consiste donc à contourner le problème de la génération des candidats, en supprimant cette notion, grâce à une structure de *FP-tree*. Cette structure, pour être mise en place, demande tout d'abord une passe sur la base de données, afin d'en collecter les items fréquents. En effet, la structure de *FP-tree*, ne considère que les items fréquents, afin d'extraire les itemsets fréquents. Une fois cette passe effectuée, la majeure partie de l'algorithme consiste à construire l'arbre, qui est une transformation de la base d'origine, limitée aux items fréquents. L'exemple 19, illustre ce concept.

Exemple 19 *Considérons la base de données représentée à la figure 3.2. La figure 3.3 représente l'arbre obtenu à partir de cette base. Tout d'abord, une passe sur la base permet de ne retenir que les items ayant un support supérieur à 3. La représentation ordonnées des itemsets est alors primordiale, dans le sens ou la construction de l'arbre va suivre cet ordre. Chaque liste (ordonnée par support décroissant) d'items fréquent donne alors lieu à un chemin dans l'arbre. Chaque chemin ainsi créé, part de la racine de l'arbre, et se termine par le dernier item de la séquence. Le premier chemin cosntruit à partir de la racine sera donc : {(f:1), (c:1), (a:1), (m:1), (p:1)}. Ensuite l'itemset (f, c, a, b, m) viendra mettre à jour une partie de ce chemin, et en créera un autre. Le support des items f, c et a est alors incrémenté de 1. Itemset par itemset, l'arbre est ainsi construit.*

L'étape finale, de la construction de cette structure, consiste à ajouter un index des sommets représentant chaque item fréquent. Ainsi l'index des sommets de l'item f, par exemple, se limite à un seul sommet, le fils $f : 4$ de la racine.
Pour extraire les itemsets fréquents, à partir de la structure *FP-tree*, les auteurs de [HPY00] proposent d'exploiter les connaissances apportées par la mise en place de l'index de la structure. La méthode consiste à trouver les itemsets fréquents qui contiennent chacun des items, en analysant ces derniers un par un. L'exemple 20 illustre cette méthode.

Exemple 20 *Considérons les itemsets qui contiennent l'item p. L'index de p dans la structure FP-tree*

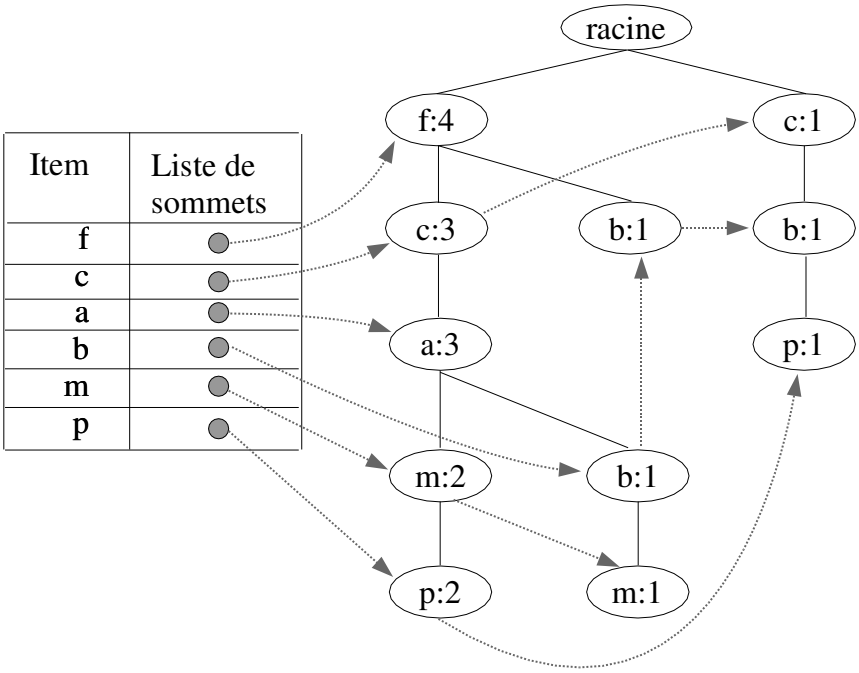

FIG. 3.3 – *La structure FP-tree après exploration de base de données représentée par la figure 3.2*

signale qu'il appartient à deux préfixes de chemin dans l'arbre. Ce sont les préfixes en question qui seront à la base de la méthode décrite. En effet le premier préfixe découvert ("$\{(f:4),(c:3),(a:3),(m:2)\}$*") indique que cet itemset suivi par p apparaît deux fois dans la base. Le second ("*$\{(f:1),(c:1),(a:1)\}$*") indique que cet itemset apparaît, suivi par p, une seule fois dans la base. La combinaison de ces deux chemins conduit à la déduction que seul c apparaît 3 fois en tant que préfixe de p. L'algorithme peut alors en déduire un seul fréquent :* (c, p) *de support 3.*

Considérons à présent l'item m. Il est également préfixé par deux chemins de l'arbre : "$\{(f:4),(c:3),(a:3)\}$*" et "*$\{(f:4),(c:3),(a:3),(b:1)\}$*". Encore une fois, la combinaison de ces deux préfixes, par un algorithme de type diviser pour régner décrit dans [HPY00], permet d'en déduire l'itemset fréquent* (f, c, a, m)

3.2 Motifs séquentiels

Elaborés en premiers lieux, les algorithmes de recherche d'associations connaissent de grandes difficultés d'adaptation aux problèmes d'extraction de motifs séquentiels. En effet, si le problème de la recherche de règles d'associations est proche de celui des motifs séquentiels (il est à son origine), les études dans ce sens montrent que, lorsque l'adaptation est possible, c'est au prix de temps de réponse inacceptables ([AS95]). Plusieurs approches, destinées à présenter une solution correcte au problème de l'extraction

de motifs séquentiels, ont été proposées depuis la définition du problème dans [AS95]. Il est important de constater que l'intérêt de la recherche de motifs séquentiels par rapport aux règles d'associations se situe à deux niveaux :

- Les motifs séquentiels sont un moyen d'obtenir une catégorie particulière de règles d'associations (intégrant une relation d'ordre entre les itemsets), par une phase de réécriture des séquences en fin de processus. Le problème de la recherche de règles d'associations est donc un sous problème de celui de l'extraction de motifs séquentiels.
- Les motifs séquentiels posent un problème plus complet et donc plus intéressant. En effet, pour définir le problème de l'extraction de motifs séquentiels, il faut prendre en compte de nouvelles contraintes apportées par la datation (voir section 2.3).

Le problème de l'extraction de motifs séquentiels est proche de celui défini par [WCM+94, RF98], situé dans la découverte de similarités dans les bases de données de séquences génétiques pour lesquelles les séquences sont des caractères consécutifs séparés par un nombre variable de caractères parasites. Dans notre définition du problème, une séquence est une liste ordonnée d'ensemble de caractères et non une liste de caractères. Si l'approche proposée par [AS95] se révèle très différente, c'est également en raison de la taille des instances traitées. En effet, les travaux de [WCM+94] sont des algorithmes basés sur des suffix-tree qui travaillent principalement en mémoire et révèlent une complexité en mémoire en $O(n)$, avec n la somme des tailles de toutes les séquences de la base. Il est impossible, dans le cadre des motifs séquentiels, d'accepter une complexité en mémoire qui dépend de la taille de la base car nous travaillons sur des instances de très grande taille et la mémoire volatile n'est jamais aussi importante que la mémoire permanente (nous ne pouvons pas non plus nous permettre de dupliquer la base ou de la réécrire sous une autre forme).

Dans [MTV95] nous sommes confrontés au problème de la découverte d'épisodes fréquents dans une séquence d'événements. S'ils utilisent la notion de durée et de time-window, leur approche est encore différente de celle définie par la recherche de motifs séquentiels généralisés. En effet leur but est de trouver les répétitions à l'intérieur d'une même séquence, en faisant se déplacer une fenêtre de temps sur cette séquence. Le problème qui nous concerne consiste, pour sa part, à rechercher des répétitions entre différentes séquences et en fonction d'une fréquence d'apparitions minimum (le support).

Dans [AMS+95] nous trouvons un résumé des techniques mises en œuvre depuis le début du projet Quest d'IBM. Ce projet est à l'origine de l'algorithme GSP, extension de Apriori, lui-même destiné à reprendre l'algorithme AIS présenté dans [AIS93]. L'algorithme GSP est présenté dans [SA96b].

3.2.1 L'algorithme GSP et sa structure

GSP (Generalized Sequential Patterns) est un algorithme basé sur la méthode générer-élaguer mise en place depuis Apriori et destinée à effectuer un nombre de passes raisonnable sur la base de données. Nous décrivons tout d'abord la méthode générer-élaguer, puis la façon dont GSP met en œuvre cette méthode ainsi que la structure de données qu'il utilise pour extraire les séquences généralisées d'une base de données.

La technique généralement utilisée par les algorithmes de recherche de séquences est basée sur une

création de candidats, suivie du test de ces candidats pour confirmer leur fréquence dans la base. Bénéficiant de propriétés relatives aux séquences et à leur fréquence d'apparition utiles à cette fin, ces techniques sont tout de même contraintes "d'essayer" des séquences avant de les déterminer fréquentes (ou non) dans la mesure où elles ne savent pas à l'avance quelles séquences vont s'avérer fréquentes. Si le coût de opérations de Lecture/Ecriture sur la base n'était pas si élevé, une technique rapide pour déterminer les fréquents consisterait à prendre un item et essayer de l'étendre le plus possible et de recommencer pour chaque item. Cela impliquerait un trop grand nombre de passes sur la base de données et donc des temps de réponses catastrophiques.
Pour permettre de garder un nombre de passes acceptable sur la base de données, la méthode générer-élaguer propose de travailler en mémoire avec les fréquents. Pour permettre cela, la méthode utilise l'algorithme générique : ALGOGENERIQUE (algorithme 3.1).

function *algoGenerique*
Input : Un support minimum σ et une base de données D.
Output : L'ensemble L des séquences ayant une fréquence d'apparitions supérieure à σ.
$k = 1$ // Les itemsets fréquents
$C_1 = \{\{< i >\}/i \in I\}$ $T = C_1$ **foreach** $d \in D$ **do** $compterSupport(C_1,\sigma,d)$ $L_1 = \{c \in C_1/Support(c) > \sigma\}$ **while** $L \neq \emptyset$ **do**
$genererCandidats(T,k)$ **foreach** $d \in D$ **do** $compterSupport(C_k,\sigma,d)$ $L_k = \{c \in C_k/Support(c) > \sigma\}$ $k = k + 1$
endwhile
return $L = \bigcup_{j=0}^{k} L_j$ **end function** *algoGenerique*

Algorithme 3.1: *Algorithme générique*

Cet algorithme utilise deux fonctions indispensables :

- COMPTERSUPPORT. Cette fonction est destinée à incrémenter le support des candidats contenus dans C_k à partir de la séquence de données d et en fonction du support minimum σ.
- GENERERCANDIDATS. Cette fonction a pour but de créer tous les k-candidats susceptibles d'être fréquents (donc tous les k-candidats susceptibles de devenir des k-fréquents) à partir d'un ensemble de (k-1)-fréquents.

Le bénéfice que cette méthode obtient, en utilisant l'algorithme ALGOGENERIQUE, se retrouve dans le fait que le nombre de passes effectuées sur la base de données à la fin du processus est exactement égal à la taille du plus grand fréquent.
Pour générer les candidats, l'algorithme GSP procède de la façon suivante : soit k la longueur des candidats à générer, plusieurs cas peuvent alors se présenter.

- $k = 1$. Pour générer les fréquents de taille 1, GSP énumère tous les items de la base et détermine en une passe lesquels ont une fréquence supérieure au support.
- $k = 2$. A partir des items fréquents, GSP génère les candidats de taille 2 de la façon suivante : pour tout couple x, y dans l'ensemble des 1-séquences fréquentes (les items fréquents), alors si $x = y$ le 2-candidat $< (x)\ (y) >$ est généré et si $x \neq y$, alors les 2-candidats $< (x)\ (y) >$ et $< (x\ y) >$ sont générés.

– $k > 2$. C_k est obtenu par auto-jointure sur L_{k-1}. La relation jointure(s_1, s_2) s'établit si la sous-séquence obtenue en supprimant le premier élément de s_1 est la même que la sous-séquence obtenue en supprimant le dernier élément de s_2. La séquence candidate obtenue par jointure(s_1, s_2) est la séquence s_1 étendue avec le dernier item de s_2. L'item ajouté fait partie du dernier itemset s'il était dans un itemset de taille supérieure à 1 dans s_2 et devient un nouvel itemset s'il était dans un itemset de taille 1 dans s_2.

Cette technique peut paraître très proche des algorithmes génétiques ([OL96, AL97]) utilisés pour écrire les heuristiques de programmation. En effet, l'algorithme procède par étapes au cours desquelles il crée des candidats en les "croisants" entre eux. Cependant il faut distinguer les méthodes de programmation par heuristiques de la méthode générer-élaguer par les différences suivantes :

– La méthode générer-élaguer n'a rien de stochastique alors que les algorithmes génétiques sont basés sur des méthodes de *roulette biaisée*.
– Les candidats génétiques forment des réponses (quelle que soit leur qualité) au problème posé, à toute étape du processus, alors que dans la méthode générer-élaguer on fait croître la taille des candidats générés jusqu'à obtenir la taille maximale.
– Généralement, les algorithmes génétiques travaillent avec un grand nombre de "candidats", car ils font leur traitement en mémoire. La méthode générer-élaguer, en revanche, génère un minimum de candidats pour limiter le nombre de tests.
– La génération des candidats dans la méthode générer-élaguer bénéficie de propriétés sur les fréquents qui lui permettent de ne pas générer des candidats dont on sait à l'avance qu'ils ne seront pas fréquents. Les algorithmes génétiques, eux, cherchent à faire au mieux pour obtenir un croisement efficace mais acceptent les dégradations.
– Les réponses fournies par une heuristique de programmation basée sur un algorithme génétique sont approximatives alors qu'un processus de fouille de données doit extraire exactement les séquences fréquentes.
– Enfin, la différence la plus importante se situe certainement au niveau du coût d'une passe sur la base. En effet, cette dernière correspondrait à la fonction objective des algorithmes génétiques qui cherchent à obtenir le meilleur compromis possible entre qualité de la fonction objective et temps nécessaire pour la mettre en œuvre.

Le théorème suivant (dont la preuve est donnée dans [SA96b]) garantit que, pour toutes les longueurs de candidats générés, si un candidat est susceptible d'être fréquent alors il sera généré.

Théorème 1 *Soit L_{k-1}, l'ensemble de (k-1)-fréquents. La génération des candidats construit un sur-ensemble de L_k, l'ensemble des k-fréquents.*

Exemple 21 *La figure 3.5 (tableau de gauche) illustre la génération des candidats de taille 2 à partir des fréquents de taille 1.*

Exemple 22 *La figure 3.5 (tableau de droite) illustre la génération des 4-candidats (colonne de droite) obtenus à partir d'un ensemble de 3-fréquents (colonne de gauche).* Nous remarquons que jointure(<(1 2) (3)>, <(2) (3 4)>) produit la séquence candidate de taille 4 <(1 2) (3 4)> et que jointure(<(1 2) (3)>, <(2) (3) (5)>) produit le 4-candidat <(1 2) (3) (5)>. Les autres séquences ne peuvent pas apparaître dans la jointure. La séquence <(1 2) (4)> ne peut pas faire partie de la relation *jointure*

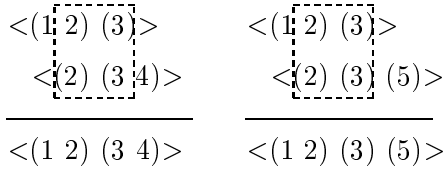

FIG. 3.4 – *2 exemples de jointure entre candidats dans GSP*

Items	2-candidats	
<(1)>	<(1 2)>	<(1) (2)>
<(2)>	<(1 3)>	<(1) (3)>
<(3)>	<(1 4)>	<(1) (4)>
<(4)>		<(1) (1)>
	<(2 1)>	<(2) (1)>
	<(2 3)>	<(2) (2)>
	<(2 4)>	<(2) (3)>
		<(2) (4)>
	<(3 1)>	<(3) (1)>
	<(3 2)>	<(3) (2)>
	<(3 4)>	<(3) (3)>
		<(3) (4)>
	<(4 1)>	<(4) (1)>
	<(4 2)>	<(4) (2)>
	<(4 3)>	<(4) (3)>
		<(4) (4)>

3-fréquents	4-candidats
<(1 2) (3)>	<(1 2) (3 4)>
<(1 2) (4)>	<(1 2) (3) (5)>
<(1) (3 4)>	
<(1 3) (5)>	
<(2) (3 4)>	
<(2) (3) (5)>	

FIG. 3.5 – *Génération des candidats dans la méthode générer-élaguer*

car il n'y a aucune séquence de la forme <(2) (4 x)>. La figure 3.4 illustre la façon dont la jointure est effectuée entre les séquences candidates.

Pour évaluer le support de chaque candidat en fonction d'une séquence de données, GSP utilise une structure d'arbre de hachage destinée à effectuer un tri sommaire des candidats.

- Les candidats sont stockés en fonction de leur préfixe, mais sans tenir compte des contraintes de temps. Pour ajouter un candidat dans l'arbre des séquences candidates, GSP parcourt ce candidat et effectue la descente correspondante dans l'arbre. En arrivant sur une feuille, GSP ajoute ce candidat à la feuille et si la taille de la feuille depasse la taille maximale alors elle est scindée en deux nouvelles feuilles dans lesquelles les candidats sont répartis.
- Pour trouver quelles séquences candidates sont incluses dans une séquence de données, GSP parcourt l'arbre en appliquant une fonction de hachage sur chaque item de la séquence de données. Quand une feuille est atteinte, elle contient des candidats potentiels pour la séquence de données. Cet ensemble de séquences candidates est constitué de candidats inclus dans la séquence de données et de candidats "parasites".

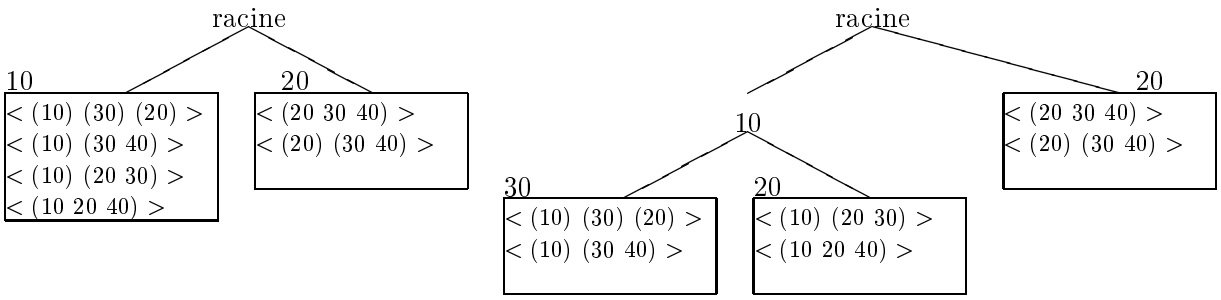

FIG. 3.6 – *La structure hash-tree de GSP*

Détaillons à présent la méthode que GSP utilise pour détecter les séquences candidates incluses dans une séquence de données. A partir d'une séquence de données d , GSP commence par explorer l'arbre à la racine en appliquant la procédure suivante, selon le nœud de l'arbre où il se trouve :

- Sur la racine : GSP applique la fonction de hachage sur chaque item de d et si la racine possède un fils correspondant à l'item alors la procédure est récursivement appliquée avec ce fils. Si une séquence candidate c est incluse dans d son premier item doit être dans d, c'est pourquoi GSP applique la fonction de hachage sur chaque item de d afin d'assurer que toute séquence qui commence avec un item qui n'est pas dans d sera ignorée.
- Sur un nœud interne (ni une feuille, ni la racine) : considérons que le nœud est atteint en hachant sur un item x de d dont la date d'achat est t. GSP applique la fonction de hachage sur chaque item de d dont la date est dans l'intervalle [t-windowSize, t+max(windowSize, maxGap)] et si le nœud considéré possède un fils correspondant à l'item alors la procédure est récursivement appliquée avec ce fils. Les détails expliquant la correction de cette méthode sont donnés dans [SA96b].
- Sur une feuille : pour toutes les séquences candidates s contenues dans la feuille GSP vérifie si d contient s. Il est possible, en effet qu'il y ait dans cette feuille une séquence non contenue dans d car GSP stocke les séquences candidates sans tenir compte des changements d'itemsets à l'intérieur des séquences candidates (cf. exemple 23).

Exemple 23 *La figure 3.6 illustre la façon dont GSP gère la structure de hash-tree.* Les deux arbres servent à conserver les mêmes candidats mais les feuilles de l'arbre de gauche peuvent contenir plus de deux séquences candidates alors que dans l'arbre de droite seulement deux séquences candidates peuvent être stockées dans la même feuille. Nous remarquons que la feuille qui porte l'étiquette 20 dans l'arbre de gauche (le deuxième fils de la racine) est utilisée pour stocker les séquences candidates $<$ (20 30 40) $>$ et $<$ (20) (30 40) $>$. Quand GSP atteint cette feuille, il n'a aucun moyen de savoir si c'est la séquence $<$ (20 30 40) $>$ ou bien la séquence $<$ (20) (30 40) $>$ qui l'a conduit jusqu'à cette feuille. C'est pour cette raison que GSP doit tester les séquences candidates contenues dans les feuilles atteintes afin de savoir quels supports incrémenter.

Pour déterminer si une séquence candidate s est incluse dans une séquence de données d GSP commute entre deux phases. GSP applique la phase *vers l'avant*, au premier élément de la séquence.

- Phase vers l'avant : l'algorithme trouve les éléments sucessifs de s dans d tant que la différence entre la date de fin de l'élement courant et la date de début de l'élement précédent est inférieure à maxGap. Si cette différence excède maxGap, l'algorithme bascule dans la phase *vers l'arrière*.
- Phase vers l'arrière : l'algorithme, au moyen d'un backtrack, "tire vers le haut" l'élément précédent. Soit s_i l'élément courant et t sa date, l'algorithme cherche le premier ensemble de transactions qui contient s_{i-1} dont les dates se situent après ($t\text{-}maxGap$).

Pour optimiser cette phase, GSP utilise une nouvelle structure de données permettant de retrouver rapidement les dates de chaque item. Cette structure est principalement utilisée par la phase vers l'arrière pour retrouver une nouvelle occurrence d'un item.

3.2.2 PrefixSpan

La méthode PrefixSpan, présentée par [PHMA+01, MA99] se base sur une étude attentive du nombre de candidats qu'un algorithme de recherche de motifs séquentiels peut avoir à produire afin de déterminer les séquences fréquentes. En effet, selon les auteurs, pour envisager d'utiliser un algorithme comme GSP il faut s'attendre à devoir générer, uniquement pour la seconde passe, pas moins de $n^2 + \frac{n \times (n-1)}{2}$ candidats de taille 2 à partir des n items trouvés fréquents lors de la première passe. L'objectif des auteurs est alors clairement affiché : réduire le nombre de candidats générés dans la mesure où il s'agit là d'un des facteurs les plus pénalisants des algorithmes de recherche de motifs séquentiels à l'heure actuelle.

Pour parvenir à cet objectif, PrefixSpan propose d'analyser les préfixes communs que présentent les séquence de données de la base à traiter. À partir de cette analyse, l'algorithme construit des bases de données intermédiaires, qui sont des projections de la base d'origine déduites à partir des préfixes relevés. Ensuite, dans chaque base obtenue, PrefixSpan cherche à faire croître la taille des motifs séquentiels découverts, en appliquant la même méthode de manière récursive.

Deux sortes de projections sont alors mises en place pour réaliser cette méthode : la projection dite "niveau par niveau" et la "bi-projection". Au final, les auteurs propose une méthode d'indexation permettant de considérer plusieurs bases virtuelles à partir d'une seule, dans le cas où les bases générées ne pourraient être maintenues en mémoire en raison de leurs tailles.

exemple de fouille avec PrefixSpan

PrefixSpan fonctionne avec une écriture de la base différente de celle utilisée par GSP. En effet cet algorithme requiert un format qui présente sur une ligne le numéro de client suivi de toutes ses transactions sous forme de séquence de données. Ce format nécessite une re-écriture de la base de données avant de procéder à l'étape de fouille de données. La base de données $DBspan$ utilisée tout au long de cet exemple est décrite par la figure 3.7.

La méthode de projection préfixée va permettre de procéder à une extraction des motifs séquentiels avec un support minimum de deux clients, en appliquant les étapes suivantes :

Client	Séquence
10	<(a) (a b c) (a c) (d) (c f) >
20	<(a d) (c) (b c) (a e) >
30	<(e f) (a b) (d f) (c) (b) >
40	<(e) (g) (a f) (c) (b) (c) >

FIG. 3.7 – *DBspan, base de données exemple pour PrefixSpan*

Préfixe	base projetée (suffixes)	motifs séquentiels
<a>	<(abc)(ac)(d)(cf)>, (_d)(c)(bc)(ae)>, <(_b)(df)(c)(b)>, <(_f)(c)(b)(c)>	<a>, <(a)(a)>, <(a)(b)>, <(a)(bc)>, <(a)(bc)(a)>, <(a)(b)(a)>, <(a)(b)(c)>, <(ab)>, <(ab)(c)>, <(ab)(d)>, <(ab)(f)>, <(ab)(d)(c), <(a)(c)(a)>, <((a)(c)(b)>, <(a)(c)(c)>, <(a)(d)>, <(a)(d)(c)>, <(a)(f)>
<b>	<(_c)(ac)(d)(cf)>, <(_c)(ae)>, <(df)(c)(b)>, <c>	<b>, <(b)(a)>, <(b)(c)>, <(bc)>, <(bc)(a)>, <(b)(d)>, <(b)(d)(c)>, <(b)(f)>
<c>	<(ac)(d)(cf)>, <(bc)(ae)>, <b>, <(b)(c)>	<c>, <(c)(a)>, <(c)(b)>, <(c)(c)>
<d>	<(cf)>, <(c)(bc)(ae)>, <(_f)(cb)>	<d>, <(d)(b)>, <(d)(c)>, (d)(c)(b)>
<e>	<(_f)(ab)(df)(c)(b)>, <(af)(c)(b)(c)>	<e>, <(e)(a)>, <(e)(a)(b)>, <(e)(a)(c)>, <(e)(a)(c)(b)>, <(e)(b)>, <(e)(b)(c)>, <(e)(c)>, <(e)(c)(b)>, <(e)(f)>, <(e)(f)(b)>, <(e)(f)(c)>, <(e)(f)(c)(b)>
<f>	<(ab)(df)(c)(b)>, <(c)(b)(c)>	<f>, <(f)(b)>, <(f)(b)(c)>, <(f)(c)>, <(f)(c)(b)>

FIG. 3.8 – *Résultat de PrefixSpan sur la base de données de la fig 3.7*

Étape 1 : Trouver les items fréquents. Pour cela, une passe sur la base de données va permettre de collecter le nombre de séquences supportant chaque item rencontré, et donc d'évaluer le support des items de la base. Les items trouvés sont (sous la forme <item>:support) : <a>:4 ,<b>:4 ,<c>:4 ,<d>:3 ,<e>:3 ,<f>:3 .
Étape 2 : Diviser l'espace de recherche. L'espace de recherche complet peut être divisé en six sous-ensembles, puisqu'il y a six préfixes de taille 1 dans la base (i.e. les six items fréquents). Ces sous-ensembles seront : (1) les motifs séquentiels ayant pour préfixe <a>, (2) ceux ayant pour préfixes <B>, ... et (6) ceux ayant pour préfixe <f>.
Étape 3 : Trouver les sous-ensembles de motifs séquentiels. Les sous-ensembles de motifs séquentiels peuvent être trouvés en construisant les projections préfixées des bases obtenues et en ré-appliquant l'algorithme de fouille de manière récursive. Les bases ainsi projetées et les motifs obtenus sont alors donnés à la figure 3.8.

Le processus de découverte des motifs séquentiels fréquents, sur les bases projetées, peut alors se dérouler de la manière suivante :

Tout d'abord, prefixSpan cherche les sous-sequences des séquences de données, ayant pour préfixe <a>. Seules les séquences contenant <a> sont à prendre en compte. De plus dans chacune de ces séquences, seul le suffixe doit être considéré. Par exemple avec la séquence <(e f) (a b) (d f) (c) (b) >, seule la sous-séquence (le suffixe) <(_b) (d f) (c) (b) > sera pris en compte (dans cette séquence le caractère "_" signifie que le préfixe était contenu dans le même itemset que "b").

Les séquences de $DBspan$ qui contiennent <a> sont alors projetées pour former $DBspan_{<a>}$, qui contient quatre suffixes : <(abc)(ac)(d)(cf)> (_d)(c)(bc)(ae)> <(_b)(df)(c)(b)> et <(_f)(c)(b)(c)>. Une passe sur $DBspan_{<a>}$ permet alors d'obtenir les motifs séquentiels de longueur 2 ayant <a> pour préfixe commun : <(a)(a)>:2, <(a)(b)>:4, <(ab)>:2, <(a)(c)>:4, <(a)(d)>:2 et <(a)(f)>:2.

De façon récursive, toutes les séquences ayant pour préfixe <a> peuvent être partitionnées en 6 sous-ensembles : (1) celles qui ont pour préfixe <(a)(a)>, (2) celles qui ont pour préfixe <(a)(b)>... et (6) celles qui ont pour préfixe <(a)(f)>. Ces motifs peuvent alors fomer de nouvelles bases projetées, et chacune de ces bases peut alors être utilisée en entrée de l'algorithme, toujours de manière récursive.

$DBspan_{<(a)(a)>}$, qui contient la projection des séquences ayant pour préfixe $< (a)(a) >$, contient une seule sous-séquence (suffixe) : <(_bc)(ac)(d)(cf)>. Comme aucune autre séquence fréquente ne peut être générée à partir d'une seule séquence (le support de 1 étant la borne inférieure), le processus quitte cette branche, pour remonter d'un niveau dans la récursivité.

$DBspan_{<(a)(b)>}$, contient trois suffixes : <(_c)(ac)(d)(cf)>, <(_c)(a)>et <c>. En fontionnant de manière récursive sur $DBspan_{<(a)(b)>}$, le processus va trouver quatre motifs séquentiels fréquents : <(_c)>, <(_c)(a)>, <a>et <c>. Après remplacement du préfixe générique ("_") par le préfixe qui a conduit à cette base projetée ("<(a)(b)>"), nous obtenons les motifs : <(a)(bc)>, <(a)(bc)(a)>, <(a)(b)(a)>et <(a)(b)(c)>.

Le processus continue alors pour les bases projetées sur <(ab)>, <(a)(c)>, <(a)(d)>et <(a)(f)>. Ensuite les autres items fréquents (b, c, d, e et f) seront examinés en tant que préfixe de sous-séquences de la base pour construire $DBspan_{<b>}$, $DBspan_{<c>}$, ... $DBspan_{<f>}$ et les analyser de manière récursive.

3.2.3 SPADE

SPADE, présenté dans [Zak99], se classe dans la catégorie des algorithmes qui, à l'instar de prefixSpan, cherchent à réduire l'espace des solutions en regroupant les motifs séquentiels par catégorie. Pour SPADE, les motifs fréquents présentent des préfixes communs, qui permettent de décomposer le problème en sous-problèmes qui seront traités en mémoire.

Le calcul de F_2 (les fréquents de taille 2) par SPADE, passe par une inversion de la base, qui la transforme d'un format vertical vers un format horizontal. Les auteurs considèrent que cette opération peut être simplifiée si la base peut-être chargée en mémoire vive. De plus, lors de leurs expérimentations, les auteurs ne considèrent pas, dans les temps de calcul relevés, le temps de re-écriture de la base, dans le cas où celle-ci ne tiendrait pas en mémoire.

Client	Itemset	Items
1	10	A B
	20	B
	30	A B
2	20	A C
	30	A B C
	50	B
3	10	A
	30	B
	40	A
4	30	A B
	40	A
	50	B

FIG. 3.9 – *DBspade, base de données exemple pour SPADE*

SPADE gère les candidats et les séquences fréquentes à l'aide de classes d'équivalence comme suit : deux k-séquences appartiennent à la même classe si elles présentent un préfixe commun de taille $(k-1)$. Plus formellement, soit $\mathcal{P}_{k-1}(\alpha)$ la séquence de taille k-1 qui préfixe la séquence α. Comme α est fréquente, $\mathcal{P}_{k-1}(\alpha) \in F_{k-1}$, avec F_{k-1} les fréquents de taille $k-1$. Une classe d'équivalence est définie de la manière suivante :

$$[\rho \in F_{k-1}] = \{\alpha \in F_k | \mathcal{P}_{k-1}(\alpha) = \rho\}$$

Chacune de ces classes d'équivalence contient alors deux types d'éléments : $[\rho.l_1] =< \rho\ (x) >$ ou bien $[\rho.l_2] =< \rho x >$, selon que l'item x appartient ou pas à la même transaction que le dernier item de ρ.

Les candidats seront ensuite générés selon trois critères :

- Auto-jointure ($[\rho.l_1] \times [\rho.l_1]$).
- Auto-jointure ($[\rho.l_2] \times [\rho.l_2]$).
- Jointure ($[\rho.l_1] \times [\rho.l_2]$).

Le reste de l'algorithme, à savoir le comptage du support pour les candidats générés, repose sur la re-écriture préalable de la base de données. En effet la transformation consiste à associer à chaque k-séquence l'ensemble des couples (client, itemset) qui lui correspondent dans la base. L'exemple 24 illustre le résultat de cette transformation, et la façon dont la table obtenue est utilisée lors du calcul du support.

Exemple 24 *La figure 3.9 représente une base de données, représentée selon le format vertical classique. Après transformation selon les besoins de l'algorithme SPADE, la base de données DBspade est alors décrite dans le cadre "B." de la figure 3.10. Une fois la base de données ainsi transformée, l'algorithme peut alors accéder aux supports des candidats de taille 2, grâce aux listes d'itemsets et clients supportant les items fréquents, en procédant à une intersection de ces listes. Considérons l'en-*

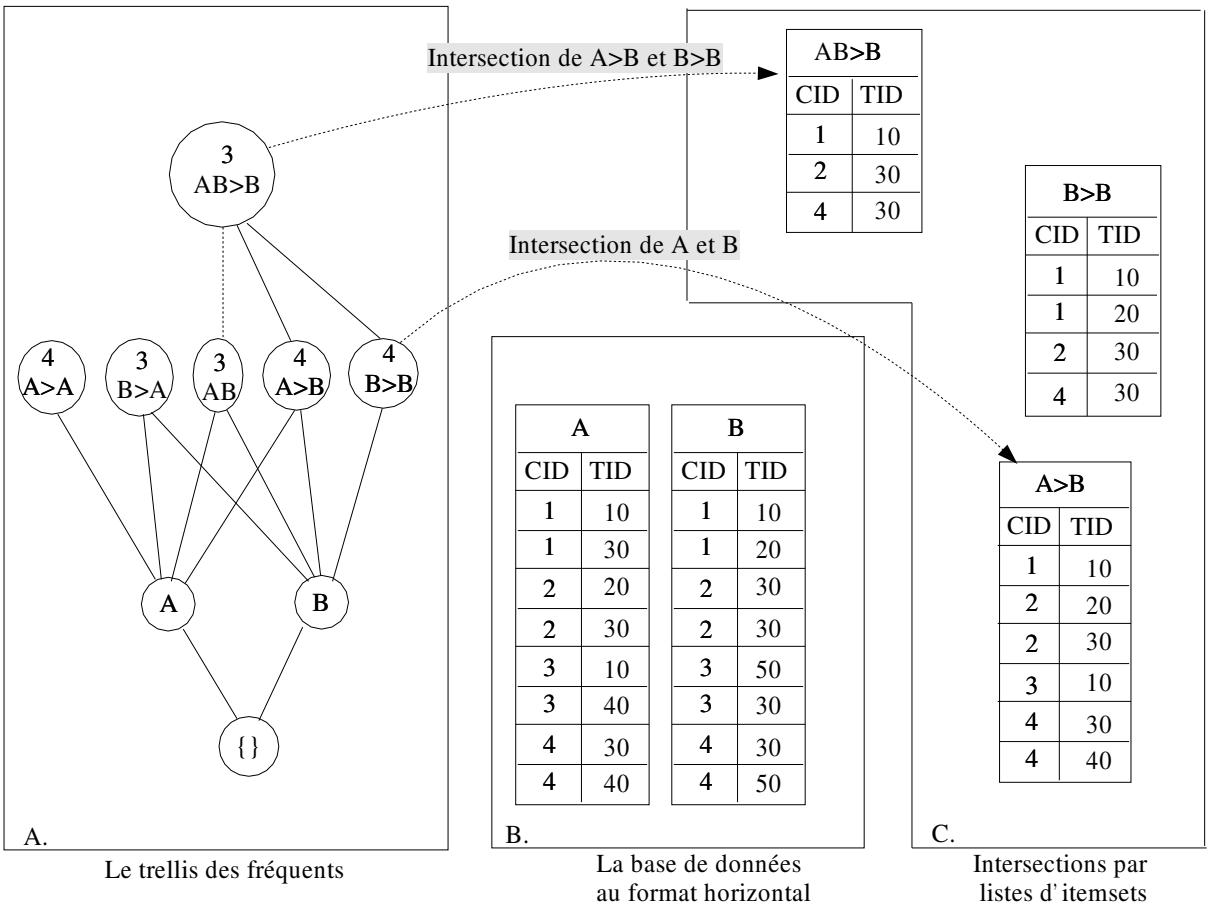

FIG. 3.10 – *Intersections de listes d'itemsets dans SPADE, avec la base de données de la figure 3.9*

chaînement "A>B" qui signifie "A est suivi par B" ou encore <(A) (B) >. En utilisant un algorithme d'intersection des listes de A et de B, SPADE peut déduire la liste d'itemsets de <(A) (B) >.

La façon dont SPADE gère les intersections est détaillée dans [Zak99].

3.2.4 Similarités avec d'autres domaines

De nombreux algorithmes performants ont été définis pour la recherche de motifs et de sous-mots dans une phrase, ainsi que la recherche de sous-mots communs. Nous pouvons citer par exemple les algorithmes Boyer-Moore ou Knuth-Morris-Pratt ([CLR94]), mais aussi plus récemment les travaux de [CR93, WM92]. Cependant ces techniques sont destinées à la localisation d'un pattern dans une phrase. Dans le problème que nous traitons, il s'agit d'exhiber des patterns pour plusieurs phrases et pour cela, de trouver quel est le nombre de phrases qui contiennent ces patterns.

Plus proches des règles d'associations, se situent certaines méthodes utilisées en IA. Nous constatons, à titre d'exemple, que [DM85] propose une méthode qui a pour but d'établir des règles sous-tendantes à la génération d'une sequence dans le but de prédire sa continuation (i.e. prédire quel est le prochain nombre dans une série de nombres). Notre problématique est différente car nous sommes intéressés par la découverte de séquences communes, que l'on peut mettre en relief dans base de données qui contient des listes d'ensembles de nombres (par opposition à la découverte de répétitions dans une liste de nombres).
Ce que nous pouvons constater, en explorant les travaux relatifs à la recherche de répétitions ou à la localisation de sous-mots, c'est la variété des domaines demandeurs pour de telles techniques : la médecine ([WCM+94, RF98]), la robotique ([YDS+96, LYS97]), les télécommunications ([QET97]) ou encore la pathologie des barrages ([CP97]), ont besoin de méthodes d'extraction de connaissances dans une base de données ou de méthodes proches de la fouille de données.

[LYS97] par exemple, propose une méthode de détection de régularités parmi un ensemble de données. Cette méthode est proche de la technique d'élaboration de règles d'associations dans la mesure où elle ne considère pas la temporalité, dimension que l'on peut affecter à l'ensemble des attributs des données traitées. Souvent, les méthodes de détection des répétitions proposées, comme par exemple celles de [Wan97, WCM+94] qui reposent sur une utilisation des suffix tree, utilisent des techniques de chargement de la base en mémoire grâce à des structures qui permettent (au prix d'une complexité en espace dépendant la taille de la phrase à analyser) de gérer très efficacement le problème de l'incrémentalité.

3.3 Recherche incrémentale de motifs séquentiels

La recherche incrémentale de règles d'association est à l'origine de nombreux travaux ([CHNW96, CLK97, AP95, SS98, TBAR97, RMR96, RMR97]), mais depuis la définition de la recherche de motifs séquentiels par [AS95], peu de travaux ont, à notre connaissance, abordé l'aspect incrémental de cette problématique, tel qu'il est présenté en introduction et défini à la section 2.4. De plus parmi ces travaux, aucun ne s'interesse à la prise en compte des contraintes temporelles ou ne semble préparé à les prendre en compte. Nous proposons dans cette section de procéder à une étude de deux points de vue existant, à l'heure actuelle, sur ce domaine : FASTUP [LL98] et une approche par SuffixTree [WT96], d'un côté, et ISM [PZOD99], d'un autre.

3.3.1 Approche par SuffixTree et FASTUP

Dans [WT96], les auteurs proposent une solution au problème de l'extraction incrémentale de motifs séquentiels, basée sur la technique des arbres postfixés (suffixtree). Nous avons vu, en introduction de ce chapitre, que la complexité en espace affichée par de telles méthodes était dépendante de la taille de la base de données à traiter. L'algorithme présenté dans cet article fait partie de ces méthodes. En revanche sa mise en place est aisée dans la mesure où il profite de la structure utilisée. En effet la structure utilisée acquiert les données et construit les fréquents en une seule passe sur la base de données en construisant l'arbre postfixé au cours de sa lecture. Cette méthode se prête donc avantageusement à la problématique de l'incrémentalité en permettant de continuer la construction de l'arbre postfixé

en "poursuivant" la lecture des nouvelles données. Une fois cette lecture achevée et l'arbre construit, les auteurs obtiennent les nouveaux fréquents.

Encore une fois, même si l'efficacité en temps d'une telle méthode ne peut qu'être reconnue, il faut être conscient de la complexité en espace d'une telle méthode. L'algorithme présenté par [WT96] (tout comme ISM présenté plus bas, dans la sous-section 3.3.2) affiche une complexité en espace dépendante de la taille de la base de données à traiter.

FASTUP, proposé par [LL98], est un exemple des premiers travaux présenté dans le domaine de la recherche incrémentale de motifs séquentiels, dont la complexité en espace dépend de la taille de la réponse et non de la taille de la base. En effet, FASTUP se veut être une sorte de GSP amélioré, tenant compte des résultats obtenus lors d'une précédente fouille de données sur la base, avant de construire, proposer et valider des candidats, selon le schéma "générer-élaguer".

FASTUP propose tout d'abord de faire une mise à jour des support pour les items, en fonction des modifications apportées par *db*, l'incrément. Pour cela, l'algorithme procède à plusieurs étapes :

- Faire une passe sur *db*,
- compter les items et leurs support à la fin de cette passe,
- confronter les résultats à ceux obtenus au préalable sur DB,
- faire une passe sur DB,
- en déduire les nouveaux items fréquents.

Au cours de la passe sur DB, FASTUP tire profit des informations collectées pour, si nécessaire, détecter les fréquents qui ne le sont plus après la mise à jour. En effet le support de certaines séquences peut baisser en cas d'ajout de clients, comme nous le détaillerons dans le chapitre 6.
Le reste de la méthode FASTUP consiste à se servir de l'ensemble de séquences fréquentes, acquis lors de l'extraction sur DB, pour éviter de générer des candidats. En effet cet algorithme repart de l'ensemble des items fréquents, et ensuite (comme le fait GSP) propose des candidats générés selon le principe "générer-élaguer", en faisant croître la taille de ses candidats de un item par génération. La différence vient du fait que, grâce au informations collectées lors de la précédente phase de fouille de données, FASTUP connaît par avance le caractère fréquent ou infréquent de certains de ses candidats et peut donc éviter la génération et la validation inutile, d'un certain nombre d'entre eux.

3.3.2 ISM

L'algorithme ISM, proposé par [PZOD99], est en fait une extension de l'algorithme SPADE, présenté dans la sous-section 3.2.3, qui prend en compte la mise à jour à l'aide d'une bordure négative et de la re-écriture de la base, implémentée par ce même algorithme.

La figure 3.11 présente la base de données $DBspade$ de la figure 3.9 après une mise à jour. On peut y constater l'ajout d'itemset sur trois clients de $DBspade$.

Client	Itemset	Items
1	10	A B
	20	B
	30	A B
	100	**A C**
2	20	A C
	30	A B C
	50	B
3	10	A
	30	B
	40	A
	110	**C**
	120	**B**
4	30	A B
	40	A
	50	B
	140	**C**

FIG. 3.11 – *DBspade, après la mise à jour*

La première exécution de SPADE sur $DBspade$, avait abouti sur le treillis donné à la figure 3.10 dans le rectangle "A.". Lors de la construction de ce treillis, SPADE a du proposer des candidats et les insérer dans le treillis, à chaque étape de l'algorithme, soit pour chaque taille j (avec $1 \leq j \leq k$ et j la taille maximale des fréquents obtenus). Ainsi le treillis a été porteur de chaque j-candidat généré.

L'idée d'ISM consiste à garder dans ce treillis la bordure négative (NB), qui est composée des j-candidats les plus bas de la hierarchie d'inclusion qui n'ont pas été retenus. En d'autres termes, soit s une séquence appartenant à NB, alors $\nexists s'/s'$ est fils de s et $s' \in NB$, et plus précisemment NB est l'ensemble des séquences qui ne sont pas fréquentes mais dont les sous-séquences qui l'ont générée sont fréquentes. La figure 3.12 donne un exemple de bordure négative (zone grisée) pour la base de données $DBSpade$ de la figure 3.9. On peut y constater que les liens en pointillés représentent une hierarchie qui n'a pas donnée lieu à un nouveau fréquent. Par exemple pour le fréquent < (A B) (B) >, seules les séquences fréquentes < (A) (B) > et < (B) (B) > lui permettent de devenir fréquent et servent à calculer sont support.

La première étape d'ISM, consiste à supprimer de l'ensemble des séquences fréquentes, celles qui ne le sont plus après la mise à jour. Une passe sur la base de données permet de régler la question et le treillis, ainsi que la bordure négative sont mis à jour pour prendre en compte le nouveau support. Au cours de étape d'ISM va déterminer les séquences de la bordure négative qui vont migrer de NB vers FS (FS désigne ici l'ensemble des séquences fréquentes). ISM obtient donc les séquences qui, en quelque sorte, vont être "dégrisées" dans le treillis. De plus, ISM met à jour l'ensemble des items fréquents à la fin de cette passe, afin d'insérer d'éventuels nouveaux items dans le treillis.

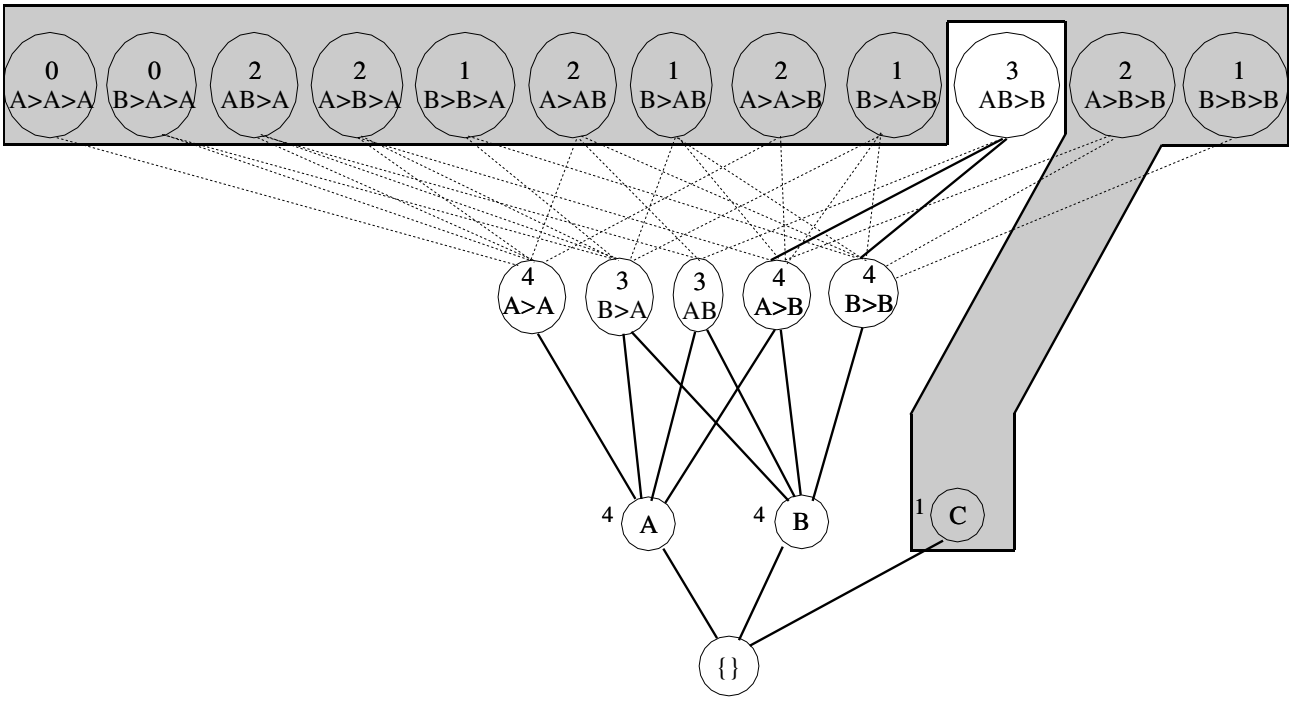

FIG. 3.12 – *La bordure négative, considérée par ISM après exécution de SPADE sur la base de données de la figure 3.9*

La seconde étape d'ISM, consiste à reprendre les nouveaux fréquents (les items ajoutés au treillis, ou les séquences qui ont migré de NB vers FS) un par un, afin de faire progresser l'information dans le treillis en reprenant le principe de génération de SPADE. Le champ d'exploration que doit considérer ISM est alors limité aux éléments nouveaux, et des optimisations sont mises en place compte tenu des caractéristiques particulières que présentent les nouveaux fréquents (pour plus de détails sur les caractéristiques que les nouveaux fréquents peuvent présenter, le lecteur peut se reporter à [PZOD99, MPT99b], ou au chapitre 6 et plus particulièrement aux lemmes et preuves de la section 6.2).

Exemple 25 *Considérons par exemple l'item "C", de la base de données DBspade. Cet item ne présente qu'un support de 1 séquence à la fin de l'algorithme SPADE. Après la mise à jour représentée à*

la figure 3.11, ISM re-considère le support de cet item, qui vaut désormais 4. "C" passe donc de NB vers FS. De la même manière, la séquence < (A) (A) (B) > et la séquence < (A) (B) (B) > deviennent fréquentes après la mise à jour et passent de NB vers FS. Toutes ces séquences sont découvertes après la première étape qui consiste à re-évaluer le support de chaque séquence appartenant à la bordure négative (NB) dans le treillis.

La seconde étape étape elle, sera destinée à reprendre les générations de candidats, mais en se limitant aux séquences qui viennent d'être ajoutées à FS. Par exemple les séquences < (A) (A) (B) > et < (A) (B) (B) >, devenues fréquentes après l'étape ci-dessus, peuvent générer le candidat > (A) (A) (B) (B) > qui sera déterminé non fréquent avec un support de 0 séquences. > (A) (A) (B) (B) > fait donc partie de la nouvelle bordure négative. Après la mise à jour représentée à la figure 3.11, l'ensemble des fréquents découverts est donc: A, B, C, < (A) (A) >, < (B) (A) >, < (A B) >, < (A) (B) >, < (B) (B) >, < (A) (C) >, < (B) (C) >, < (A) (A) (B) >, < (A B) (B) >,< (A) (B) (B) >, < (A) (A) (C) >, < (A) (B) (C) > .

À la fin de cette dernière étape, le treillis est mis à jour et ISM peut alors présenter le nouvel ensemble de séquences fréquentes, ainsi qu'une nouvelle bordure négative, permettant à l'algorithme de prendre en compte une nouvelle mise à jour.

3.4 Recherche de structures fréquentes

La frontière qui sépare la recherche de structures fréquentes et la recherche de motifs séquentiels est assez floue pour permettre de classer certains travaux dans l'une et l'autres de ces deux catégories. L'aspect le plus distinctif se situe au niveau de la problématique qui peut accepter une réduction vers le problème de la recherche de motifs séquentiels. [WL97b] présente des travaux qui sont justement à la frontière entre les deux domaines avec la recherche de "transactions emboitées". Le but de la fouille de transactions emboitées est de trouver toutes les "collections" qui sont une généralisation d'un pourcentage minimum de transaction, et que l'on appelle *motifs d'association imbriqués*. Les travaux qu'il présente gèrent une variante de la notion d'inclusion avec le concept "weaker than" (moins fort que). Ce concept est défini de la manière suivante :

$$T_1 x_1, x_2, \ldots x_n \text{ weaker than } T_2 x'_1, x'_2, \ldots x'_n \text{ si } \forall x_i \in T_1, \exists x_j \in T_2 \; x_i \subseteq x_j.$$

À partir de la définition de "weaker than", la fouille de transactions emboitées consiste à trouver toutes les transactions s telles que le nombre de transactions "plus faibles" que s soit supérieur au support minimum.

A notre connaissance, il existe très peu de travaux concernant la recherche de régularités structurelles dans de grandes bases de données. Néanmoins, notre façon d'aborder le problème est très proche de celle proposée dans [WL98, WL99] pour la recherche d'associations structurelles dans des données semi-structurées. Les auteurs proposent une approche très efficace et des solutions basées sur une nouvelle représentation de l'espace de recherche. En outre, en proposant des optimisations basées sur des stratégies d'élagage, ils améliorent considérablement l'étape de génération des candidats.

L'approche WARMR proposée dans [DTk98] aborde la recherche de sous structures fréquentes dans des

composants chimiques. Toutefois même si la problématique est similaire, ils considèrent que les sous-structures recherchées sont exprimées sous la forme de requête DATALOG. De manière générale, les motifs autorisés sont spécifiés via un langage déclaratif (de la même manière que les approches basées sur la logique inductive (ILP)) et sont stockés dans un treillis. A partir d'un algorithme par niveau ils déterminent quelles sont les clauses les plus fréquentes et trouvent ainsi les structures fréquentes.

Dans [HF95, SA95] des approches pour rechercher des règles d'association multi-niveaux ont été proposées. Les auteurs supposent que pour rechercher de telles règles, ils disposent d'une base de données de transactions de clients et d'une taxonomie (*hiérarchie is-a*) entre les items achetés par les clients. Les travaux proposés semblent proches de notre problématique dans la mesure où les éléments recherchés sont des ensembles d'items issus de différents niveaux de taxonomie.

Enfin, la découverte d'information structurelle à partir de données semi-structurées a été très largement étudiée. Dans ce contexte, il existe un grand nombre de propositions pour extraire les types sous-jacents de données semi-structurées [NUWC97, BDHS97, NAM98]. Par exemple, dans [NAM98], les auteurs se sont intéressés à l'extraction de la structure implicite d'un schéma semi-structuré.

3.5 Web Usage Mining

Un panorama des systèmes et approches de Web Usage Mining ainsi que de leur utilisation est proposé dans [Coo00]. Nous nous intéressons dans cette partie, aux approches répondant à la problématique présentée.

Une approche pour découvrir des informations à partir de fichiers access log est présentée dans [MJHS96, CMS97]. Les auteurs proposent l'architecture d'un système pour le Web Mining appelée WEBMINER. Même si les contraintes de temps ne sont pas prises en compte par le système, une approche pour rechercher des motifs séquentiels est proposée. Dans ce cas, le fichier access log est réécrit de manière à regrouper toutes les transactions d'un utilisateur si elles sont suffisamment proches dans le temps. Une transaction temporelle est alors vue comme un ensemble d'URL et de temps d'accès tels que les entrées sont regroupées si elles s'inscrivent dans un intervalle de temps Δt précisé par l'utilisateur. Un algorithme de recherche de règles d'association, similaire à celui de [AS94], est adapté aux motifs séquentiels. Enfin, le système propose un langage d'interrogation, basé sur SQL, pour offrir un meilleur contrôle sur le processus d'extraction.
Dans [MTV97], un algorithme efficace pour la recherche de séquences d'événements, MINEPI, est utilisé pour extraire des règles à partir du fichier access log de l'Université d'Helsinki. Chaque page consultée est considérée comme un événement et une fenêtre de temps similaire au paramètre Δt de [CMS97] permet de regrouper les entrées suffisamment proches.
Dans le projet WebLogMiner, [ZXH98], les auteurs proposent d'utiliser un système OLAP (On-Line Analytical Processing) pour extraire des informations significatives. Les données sont tout d'abord filtrées pour éliminer les informations non pertinentes puis stockées dans une base de données relationnelle. Ensuite, une structure de tableau multidimensionnel, appelée *Web Log data cube*, est construite

et chaque dimension représente un champ avec toutes les valeurs possibles décrites par les attributs. Le système OLAP est alors utilisé dans la troisième phase pour appliquer des opérations de *drill-down*, *roll-up*, *slice* et *dice* sur le data cube. Ces opérations offrent ainsi la possibilité d'examiner les données sous différents angles. Enfin, des fonctions de fouille de données comme la caractérisation, la recherche de règles d'association, la prédiction ou la classification peuvent être utilisées sur le Web Log data cube. Une approche similaire est présentée dans [Dyr97].

3.6 Discussion

Nous avons pu constater, au cours de cette exploration des principales méthodes d'extraction d'itemsets fréquents ou de motifs séquentiels, que le principal problème reste le nombre de candidats à tester. Pour une méthode basée sur la lecture de la base en mémoire, telle que celle qui exploite la structure *FP-tree*, le nombre de candidats est réduit à néant mais la complexité en mémoire dépend de la taille de la base. Pour d'autres, comme GSP, qui considèrent que la base ne peut être chargée (même re-écrite) en mémoire, la complexité en temps peut alors devenir un obstacle, en raison d'un nombre de candidats, qu'il faudra tester sur la base, parfois trop grand. La première partie de cette discussion a donc pour but de faire le point sur la question de l'équilibre à trouver entre le nombre de candidats à génerer d'un côté, et la gestion de la mémoire de l'autre.
Le deuxième aspect de cette discussion concerne les types d'applications qui peuvent être réalisées à partir de l'extraction de motifs séquentiels. Nous y abordons les problèmes de précision des résultats obtenus par les méthodes existantes, mais aussi le manque de qualité dont ces résultats peuvent faire preuve.

3.6.1 Algorithmes : génération des candidats contre complexité en mémoire

prefixSpan

Le format de données utilisé par prefixSpan lui confère une efficacité sur la manipulation de préfixes et de suffixes des séquences contenues dans la base, tout en empêchant la prise en compte d'un facteur important : les contraintes de temps. En effet la seule information liée à la temporalité que l'on retrouve dans ce format consiste à limiter les itemsets, et donc à fournir un ordre dans les événements. Cela revient effectivement à une notion de séquence, mais l'aspect temporel est limité au strict minimum et une évolution de prefixSpan vers un algorithme d'extraction de motifs séquentiels généralisés (cf section 2.3), peut s'avérer complexe.
Une autre difficulté rencontrée par prefixSpan, et pointée du doigt par les auteurs, se situe au niveau de l'écriture des bases projetées par les imbrications récursives de l'algorithme. En effet, ces bases ont une intersection non vide et contiennent au total bien plus de transactions que la base d'origine. Si la place disponible est suffisante, les auteurs affirment que le temps nécessaire à la re-écriture de la base en plusieurs bases projetées est compensé par le fait que les motifs séquentiels sont trouvés en fin de course de chaque branche de la récursivité mise en place. Cependant, si la base ne tient pas dans l'espace mémoire disponible, les auteurs proposent différentes solutions qui vont de la projection "bi-niveaux" à la gestion virtuelle par index de bases de données intermédiaires. Ces solutions sont également gourmandes en mémoire et un bilan des avantages et inconvénients de ces techniques est

proposé dans [MA99].

SPADE

De la même façon que pour prefixSpan, SPADE requiert une grande disponibilité en mémoire pour reformuler l'écriture d'origine de la base de données traitée. En effet pour chaque motif séquentiel découvert, SPADE lui associe l'ensemble des couples (client, itemsets) pour lesquels ce motif apparaît. Il en resulte des temps de calcul performants au moment de déduire les supports des candidats générés à chaque étape, mais au détriment d'une occupation exagérée de l'espace mémoire. Même si les auteurs précisent qu'une optimisation sur la gestion de la mémoire consiste à ne garder que les classes d'équivalence des fréquents de longueur $k-1$ et k, ils posent pour postulat de départ que les tables générées peuvent être accueillies en mémoire. Dans le cas où l'espace mémoire serait insuffisant, les auteurs proposent de sauver sur disque les tables de chaque classe pour les traiter de manière individuelle. Cependant, [Zak99] ne propose pas d'expérimentation mettant en jeu cette re-écriture des bases sur disque qui demande des temps d'accès en écriture, puis en lecture (car il faut que l'algorithme consulte les classes d'équivalence générées afin d'en tirer profit lors de l'intersection des listes d'itemsets).

GSP

Depuis la première apparition du problème de l'extraction de règles d'associations en fouille de données, avec l'algorithme AIS, les travaux dédiés à cette problématique ont connu des progrès considérables. La mise au point, par exemple de la méthode générer-élaguer pour l'algorithme Apriori est peu remise en question et reste la base d'un grand nombre d'algorithmes. Il est intéressant de constater que cette méthode, issue de la volonté d'obtenir un bon compromis entre opérations CPU et accès disques, se prête aussi bien aux problèmes d'extraction de séquences. En effet, l'algorithme GSP repose lui aussi sur les croisements de fréquents et le test de candidats afin de trouver les séquences fréquentes maximales dans une base de données.
Parmi les idées qui contribuent à faire de GSP l'algorithme référence en matière d'extraction de motifs séquentiels, nous pouvons également compter :

- La séparation des candidats. En créant des partitions dans l'ensemble des candidats, GSP peut éviter un certain nombre de vérification car il crée des partitions de candidats dont il sait à coup sûr que ces candidats présentent un préfixe commun (hors datation) qui les rend inaccessibles pour telle ou telle séquence de données.
- L'organisation des séquences candidates ainsi partitionnées dans un arbre. L'arbre de hachage utilisé par GSP semble être un moyen efficace de parcourir l'ensemble des candidats.

Cependant, après avoir proposé un arbre et le parcours qui lui est associé, [SA96b] limite cet arbre à une mise en commun de préfixes indépendante de la temporalité du problème (la structure ne fait aucune distinction entre deux items du même itemset et deux items avec changement d'itemset). De ce fait, GSP se voit contraint, lorsqu'il atteint une séquence, de tester si les dates des items qu'elle représente sont homogènes pour la séquence de données testée. En factorisant les séquences candidates selon leur préfixes et en tenant compte de la temporalité, GSP pourrait éviter ce test en fin de parcours et garder une structure d'arbre efficace. C'est cette factorisation sur les préfixes lors du parcours et de la gestion de l'arbre des candidats que nous proposons dans le chapitre 4.

3.6.2 Applications : de la précision et de la qualité des résultats

La recherche de résularités structurelles

Nous avons vu que l'approche de [WL98, WL99] présentait des points communs avec la problématique que nous abordons. Notre approche a cependant un certain nombre de différences avec cette dernière. Nous nous intéressons à toutes les structures stockées dans les transactions de la base alors qu'ils sont uniquement intéressés par la recherche de *tree expression* qui sont définis comme un chemin allant de la racine d'un graphe OEM à un nœud final du graphe. Avec cette définition de *tree expression*, ils ne peuvent pas trouver de régularités de la forme *identity*: {*address*: < *street*: $\perp$, *zipcode*: $\perp$ >} qui seraient incluses dans une transaction plus longue mais qui elle n'est pas fréquente. En fait, lors du parcours de la base de données pour rechercher les structures fréquentes, ils ne recherchent que les arbres maximaux et quand seulement une partie de cet arbre est fréquente, elle n'est pas découverte.

Les approches de [HF95, SA95] sont très différentes de la notre car elles s'intéressent uniquement à des données enrichies par une hiérarchie is-a alors que nous nous intéressons à des transactions dans lesquelles les données sont enrichies aussi bien par des hiérarchies *ensemble-de* que *liste-de*.

Enfin, nous avons vu que [NAM98] abordait l'extraction de types sous-jacents dans les données semi-structurées. Ces approches sont cependant éloignées de notre problématique dans la mesure où nous nous intéressons à la répétition de structures dans un schéma. Par contre, dans notre contexte, de telles approches peuvent être très utiles pour réaliser une étape de pré-traitement des données afin de transtyper les données d'un graphe OEM. Dans ce cas, nous pourrions compléter la recherche de régularités en enrichissant les éléments manipulés avec les types de données associés.

L'analyse du comportement des utilisateurs d'un site web

Parmi les approche que nous avons présentées, certaines font appel à la recherche de règles d'association [MJHS96]. Le type de comportements fournis par un algorithme de règles d'association, dans le cadre de l'analyse de comportements des utilisateurs d'un site web, nous paraît difficilement adapté pour cette problématique. Les items, manipulés par les algorithmes de recherche de règles d'association, sont traités dans un seul itemset. Les règles d'association ne peuvent donc fournir que des comportements du type `(index.html panier.html)` $\rightarrow$ `(article.html)`. Cette règle sera déduite de l'itemset `(index.html panier.html article.html)`, grâce aux supports des itemsets `(index.html panier.html)` et `(article.html)`. Si l'extraction de ces itemsets peut fournir une information intéressante, elle peut aussi totalement inverser la réalité des comportements à extraire. En effet un algorithme d'extraction de motifs séquentiel, exploité sur le même fichier log, sera capable de fournir l'information suivante : < `(index.html) (article.html) (panier.html)` >. Ce comportement est alors plus riche en information, grâce à la prise en compte de la temporalité, qui a ordonnée les items et placé la page de l'article avant la page du panier (payement par CB sur le site). Il était en effet illogique de penser que le payement (page du panier) se fasse avant que le l'utilisateur ne consulte la page de l'article (comme l'itemset `(index.html panier.html article.html)` aurait pu le laisser penser).

Les méthodes, que nous avons présentées ensuite, se sont intéressées à différents algorithmes de fouille de données, pour l'analyse du comportement d'utilisateurs d'un site web. Ces approches se classent dans la même catégorie que notre première contribution, présentée dans ce mémoire. Cependant, nous avons voulu dépasser le concept actuel de Web Usage Mining, en lui apportant les notions qui nous semblaient lui faire défaut. En effet, dans une analyse sur la précision et la qualité des résultats, nous avons remarqué deux lacunes aux techniques de Web Usage Mining basées sur la recherche de motifs séquentiels :

- Le niveau de précision des résultats. En effet, les résultats obtenus tiennent compte du comportement de l'utilisateur sur le site web concerné (et ce site seulement). Le contexte actuel du web permet d'avoir plus d'informations sur la navigation de l'utilisateur que celles qui se limitent au site traité. Utiliser ces informations peut permettre de mieux analyser les comportements.
- L'adéquation entre les résultats et les comportements réels. Nous pensons que, pour adapter le site à la navigation de l'utilisateur, il n'est pas nécessaire de prendre en compte les règles obtenues par l'analyse du comportement de tous les autres. Tenter des prédictions sur le comportement d'un utilisateur connecté la nuit depuis un pays qui est en décalage de plusieurs heures, en fonction du comportement des utilisateurs du site (toutes connections confondues), nous paraît être un exemple de Web Usage Mining approximatif. Il existe encore, dans l'utilisation des fichiers logs ou de l'architecture du web, un potentiel peu exploité qui promet pourtant des résultats d'une qualité indispensable.

Deuxième partie

Motifs séquentiels : Algorithmes et Applications

Chapitre 4

Motifs séquentiels

La structure de données utilisée par GSP nous semble mal adaptée à la méthode générer-élaguer utilisée par la majorité des algorithmes de recherche de séquences depuis AprioriAll. Nous justifions cette opinion par deux arguments principaux basés sur l'espace mémoire nécessaire à une telle structure mais également sur le manque d'efficacité d'un parcours (aussi optimisé soit-il) utilisant cette structure. Nous définissons donc une nouvelle structure, destinée à accélérer les processus de vérification des séquences candidates. Nous décrivons cette structure de données, inspirée de [Mue95], dans la section 4.1. Cette structure implique, bien sûr, de revoir la méthode de vérification des candidats. L'algorithme PSP (Prefix-tree for Sequential Patterns) [MCP98], destiné à parcourir l'arbre des séquences candidates est décrit dans la section 4.2. La section 4.3 propose différentes optimisations qui ont permis d'accélérer la navigation au sein de l'arbre des candidats. Nous présentons les différents jeux d'essais effectués avec des données réelles et les benchmarks proposés par [AS95] dans la section 4.4. Enfin dans la partie discussion, nous revenons sur la comparaison des structures de données mises en jeu par GSP et PSP.

4.1 La structure *prefix-tree*

La différence principale, distinguant notre structure de celle utilisée dans [SA96b], vient du fait que chaque chemin de la racine vers une feuille de l'arbre représente une séquence complète et une seule. Ainsi, pour des séquences candidates de longueur k (i.e. les k-candidats), la profondeur de l'arbre est exactement égale à k. D'autres caractéristiques de cette structure contribuent à la distinguer de l'arbre de hachage utilisé par [SA96b] et plus particulièrement le fait que les candidats et les séquences fréquentes de longueur $(0,\ldots,k)$ sont gérés par une structure unique. Dans la suite de ce chapitre, nous utilisons la base de données illustrée par la figure 4.1 pour décrire la structure et expliquer le mécanisme de génération des candidats.

Client	Date	Items
C1	01/01/1998	10,30,40
C1	02/02/1998	20,30
C2	11/01/1998	10
C2	12/01/1998	30,60
C2	23/01/1998	20,50
C3	01/01/1998	10,70
C3	12/01/1998	30
C3	15/01/1998	20,30

FIG. 4.1 – *Base de données exemple*

4.1.1 Description

Nous commencons par décrire la façon dont sont stockés les items fréquents ($k = 1$) puis nous précisons le fonctionnement de la structure pour $k > 1$.

k = 1

Chaque branche issue de la racine relie celle-ci à une feuille qui représente un item. Chacune de

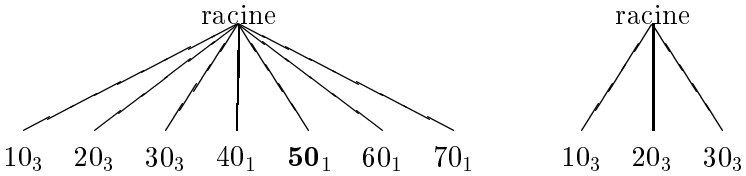

FIG. 4.2 – *Phase de pruning sur les 1-itemsets*

ces feuilles contient l'item et son support (le nombre de ses apparitions dans la base). Ce support est calculé par un algorithme comptant le nombre d'occurrences de l'item dans la base et dans des séquences distinctes (en accord avec la remarque 1, page 51).

Exemple 26 *L'arbre représenté à gauche de la figure 4.2 illustre l'état de la structure après l'évaluation du support pour chaque item.* Considérons un support de 60%, i.e. pour qu'une séquence soit retenue elle doit apparaître dans au moins 2 séquences de données. L'arbre de droite pour cette même figure représente la structure contenant uniquement les items fréquents. Considérons la feuille contenant le sommet **50** dans l'arbre de gauche, son support est de 1 séquence, ce qui signifie que seulement 33% des séquences contiennent cet item. Cet item ayant un support inférieur aux 60% spécifiés par l'utilisateur, il est éliminé des séquences candidates lors de la phase d'élagage.

k > 1

Chaque nœud de l'arbre représente un item pour une ou plusieurs séquences. Chaque chemin de la racine de l'arbre vers une feuille représente une séquence. Pour distinguer les itemsets à l'intérieur d'une séquence (ex. (10 20) et (10) (20)) nous avons séparés les fils d'un nœud en deux catégories : "Same Transaction" et "Other Transaction".

En outre, chaque feuille possède un identifiant de la dernière transaction ayant participé à l'incrémentation de son support (*idLast*). Il se peut, en effet, que l'algorithme de vérification des candidats passe plusieurs fois par une même feuille quand le chemin menant de la racine vers cette feuille est inclus plusieurs fois dans la séquence de données considérée. Dans un tel cas, la séquence candidate n fois incluses ne peut être considérée comme n fois présente dans la base uniquement à cause de cette séquence de données. L'algorithme de vérification des candidats peut ainsi savoir si la séquence de données qu'il teste a déjà participé à l'incrémentation du support de la feuille considérée.

Exemple 27 *L'arbre de la figure 4.3 représente les séquences fréquentes de tailles 1 et 2.* Une branche en traits pleins marque le début d'un nouvel itemset dans la séquence (dans la séquence < (10) (30) >, il y a une branche en trait plein entre les items 10 et 30) et une branche en traits pointillés entre deux items signifie que ces items font partie de la même transaction (< (20 30) >). Les séquences fréquentes représentées sont donc : {< (10) (30) > , < (10) (20) > , < (20 30) > , < (30) (20) >}.

Exemple 28 *L'arbre représenté par la figure 4.4 illustre la façon dont les k-candidats et les l-séquences fréquentes (avec $l \in [1..(k\text{-}1)]$) sont simultanément gérés par la structure.* Cet arbre est obtenu après la génération des candidats de taille 3 à partir de l'arbre représenté par la figure 4.3. Le mécanisme de génération de ces candidats est détaillé dans la section 4.1.2.

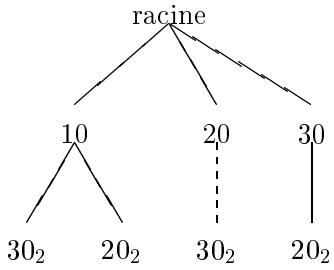

FIG. 4.3 – *Séquences fréquentes de tailles 1 et 2*

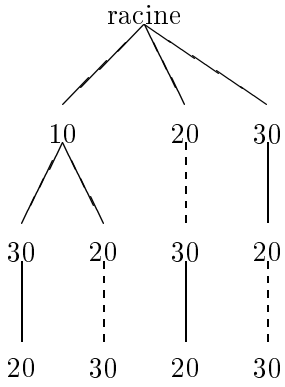

FIG. 4.4 – *Les 3-candidats obtenus avec la base exemple*

On remarque que les séquences fréquentes obtenues à partir de cet exemple sont < (10) (30) (20) >, < (10) (20 30) > et < (30) (20 30) >.

La propriété 3 garantit que la structure proposée offre un comportement en adéquation avec la définition du problème. La proposition 1, nous assure du profit apporté par cette structure quand à l'espace mémoire requis.

propriété 3 *(reformulation de la propriété 1, page 52)*
Sur un chemin de l'arbre, la valeur du support augmente d'un nœud à l'autre. Plus formellement: $\forall x \in T\, \forall y \in chemin(racine,x)$, $y.support < x.support$.

Reprenons l'arbre illustré par la figure 4.4 et considérons la séquence < (10) (30) (20) > représentée par le chemin le plus à gauche. La séquence < (10) (30) > ne peut pas avoir un support inférieur à la séquence < (10) (30) (20) > d'après la propriété 1, ce qui implique que le nœud 20 a un support inférieur au support du nœud 30 sur ce chemin.

proposition 1 *La structure arborescente présentée nécessite moins d'espace mémoire en comparaison de celle utilisée par [SA96b].*

Cette proposition se justifie de manière triviale. En effet, notre structure stocke des séquences en factorisant les préfixes de manière à ne pas les dupliquer d'une séquence à l'autre, permettant ainsi de consommer moins de mémoire qu'une méthode visant à stocker explicitement et entièrement chaque séquence de manière distincte. D'une part, dans la structure utilisée par GSP, chaque feuille contient un ensemble de séquences, donc toutes les séquences sont entièrement stockées et les préfixes communs ne sont pas exploités, mais d'autre part cette structure conserve également des chemins permettant d'accéder à ces feuilles.

4.1.2 Génération des candidats

Pour générer les candidats de longueur 1 et 2, qui constituent des cas particuliers, nous adaptons notre méthode afin de correspondre à la génération des candidats décrite dans [SA96b]. Nous remarquons de plus que les candidats générés par notre algorithme (et conservés par notre structure) y sont identiques pour toute longueur de candidats. Nous allons donc analyser la génération des candidats en fonction de k, leur longueur.

k $\in$ [**1**,**2**]
Les candidats de longueur 1 constituent l'ensemble des items de la base. Il sont donc générés de façon directe en parcourant l'ensemble des séquences de données. Les candidats de longueur 2 sont générés comme suit : pour tout couple x, y dans l'ensemble des 1-séquences fréquentes (les items fréquents), alors si $x = y$ nous générons le 2-candidat < (x) (y) > (car < $(x\ x)$ > ne peut pas faire partie de la base d'après les définitions proposées par [SA96b]) et si $x \neq y$ nous générons les 2-candidats < (x) (y) > et < $(x\ y)$ >. L'exemple 29 illustre cette phase de génération des candidats.

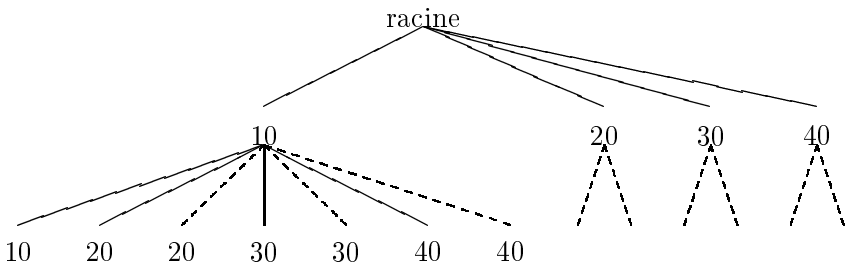

FIG. 4.5 – *Séquences candidates de taille 2*

Exemple 29 *Considérons l'arbre réduit aux fréquents de taille 1, représenté par la figure 4.2 (arbre de droite).* Les candidats de taille 2 (les 2-candidats) obtenus à partir de cet arbre sont alors illustrés par la figure 4.5 (seule la façon dont le nœud 10 est étendu est illustrée, cependant la technique est la même pour les autres nœuds).

Exemple 30 *L'arbre représenté par la figure 4.4 représente les 3-candidats obtenus depuis l'arbre des fréquents illustré par la figure 4.3.* Ces 3-candidats seront obtenus pour la base donnée exemple donnée à la figure 4.1).

k > 2

L'algorithme CANDIDATEGENERATION (cf. algorithme 4.1) de génération des séquences candidates construit l'ensemble L des feuilles de profondeur (k-1). Pour chacune de ces feuilles l, l'algorithme recherche à la racine l'item x représenté par l. Ensuite l'algorithme étend la feuille l en construisant pour cette feuille une copie des fils de x. A cette étape de la génération, l'algorithme applique un filtrage destiné à ne pas générer de séquences dont nous savons à l'avance qu'elles ne sont pas fréquentes. Pour cela, l'algorithme considère F, l'ensemble des fils de x. Pour chaque f dans F, si f n'est pas frère de l alors il est inutile d'étendre l avec f. En effet, nous savons que si f n'est pas frère de l, alors soit p le père de l, (p,f) n'est pas fréquent et donc (p,l,f) non plus. L'exemple 31 illustre l'emploi de cette optimisation.

Exemple 31 *La figure 4.6 représente un arbre avant (arbre $\mathcal{A}$) et après (arbre $\mathcal{B}$) la génération des candidats de taille 3.* La feuille représentant l'item 2 (en gras dans l'arbre $\mathcal{A}$) est étendue (dans l'arbre $\mathcal{B}$) uniquement avec les items 3 et 4 (pourtant 5 est un fils de 2 à la racine). En effet, 5 n'est pas un frère de 2 (en gras dans l'arbre $\mathcal{A}$), ce qui signifie que <(1)(5)> n'est pas une séquence fréquente. Donc, d'après la propriété 2 (page 52), <(1)(2)(5)> ne peut pas être déterminé fréquent : il est inutile de générer ce candidat.

Le théorème 2 garantit que les candidats générés par CANDIDATEGENERATION, en utilisant notre structure, sont identiques aux candidats générés par [SA96b].

Théorème 2 *Quelle que soit la longueur k des candidats générés, l'ensemble des candidats représenté par la structure prefix-tree est équivalent à l'ensemble des candidats générés suivant la description faite par [SA96b].*

function *candidateGeneration*
Input : L'arbre des candidats T, de profondeur k ($k > 2$) représentant les h-fréquents ($h \in [1..k]$).
Output : T étendu à la profondeur k+1, contenant les candidats de longueur k+1 à tester.
if $k = 2$ **then**
  foreach $n \in root.children$ **do**
    foreach $n_{add} \in root.children$ **do**
      if $n = n_{add}$ **then**
        $n.other = n$
      else
        $n.other = n_{add}$ $n.same = n_{add}$
      endif
    endforeach
    $init\_added\_leaves$ *with* $cpt = 0$ *and* $idLast = -1$
  endforeach
else
  $N_T = \{N \in T / leaf(N)$ *and* $N.depth = k\}$ /* toutes les feuilles de profondeur k */
  foreach $N \in N_T$ **do**
    foreach $N_r \in root.children$ **do**
      if $|N| = |N_r|$ *and* $N_r.child \in N.brother$ **then**
        $N.children = N_r.children \cap N.brother$
      endif
    endforeach
    foreach $n \in N.children$ **do**
      $n.cpt = 0$ $n.idLast = -1$
    endforeach
  endforeach
endif
end function *candidateGeneration*

Algorithme 4.1: *Génération des candidats de longueur k+1*

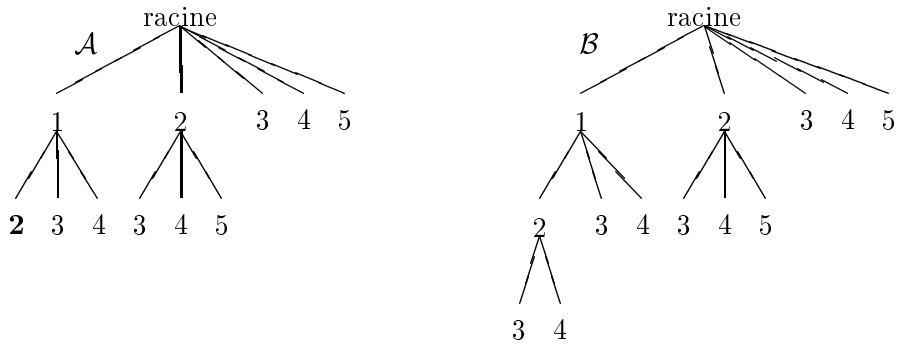

FIG. 4.6 – *Un candidat non fréquent détecté à l'avance*

Démonstration
Soit A l'arbre généré par la méthode GSP et B l'arbre généré par CANDIDATEGENERATION. Pour montrer que $A \equiv B$ quelle que soit k la longueur des candidats nous utilisons deux preuves par induction (pour $A \subseteq B$ et $B \subseteq A$).

A $\subseteq$ B
Nous voulons montrer que $\forall k\ A_k \subseteq B_k$, par induction sur k :

k=1 et k=2 sont des cas spéciaux pour lesquels l'équivalence est forçée par construction.
k=3 : soit $S_k \in A_k = (i_1,i_2,i_3)$. S_k est obtenue car $\exists S^1_{k-1} \in A_{k-1}$ et $S^2_{k-1} \in A_{k-1}$ telles que $S^1_{k-1} = (i_1,i_2)$ *et* $S^2_{k-1} = (i_2,i_3)$. Mais si (i_1,i_2) et $(i_2,i_3) \in A_2$ alors (i_1,i_2) et $(i_2,i_3) \in B_2$ (voir k=2) et par construction i_2 sera étendu avec i_3 pour obtenir $(i_1,i_2,i_3) \in B_3$.

$k \geq 3$: Pour montrer que $A_k \subseteq B_k \Rightarrow A_{k+1} \subseteq B_{k+1}$ nous utilisons l'hypothèse d'induction suivante : $\forall i \in [0..k]$, $A_i \subseteq B_i$. Soit $S_{k+1} \in A_{k+1} = (i_1,i_2,\ldots,i_{k-1},i_k,i_{k+1})$ nous devons avoir : $\exists S^1_k \in A_k = (i_1,i_2,\ldots,i_{k-1},i_k)$ et $\exists S^2_k \in A_k = (i_2,\ldots,i_{k-1},i_k,i_{k+1})$.
Si $\exists S^2_k \in A_k$ alors nous avons : $\exists S^1_{k-1} \in A_{k-1} = (i_2,\ldots,i_{k-1},i_k)$ et $\exists S^2_{k-1} \in A_{k-1} = (i_3,\ldots,i_{k-1},i_k,i_{k+1})$
$\vdots$
Nous pouvons ainsi remonter jusqu'à : $\exists S^2_2 = (i_k,ik+1) \in A_2$.
Or d'après l'hypothèse d'induction $A_2 \subseteq B_2 \Rightarrow S^2_2 = (i_k,ik+1) \in B_2$ et $S^1_k = (i_1,i_2,\ldots,i_{k-1},i_k) \in B_k$.
Donc, par construction de B, nous avons : $S_{k+1} = (i_1,i_2,\ldots,i_{k-1},i_k,i_{k+1}) \in B_{k+1}$.

B $\subseteq$ A
Nous voulons montrer que $\forall k\ B_k \subseteq A_k$, par induction sur k :

k=1 et k=2 sont des cas spéciaux pour lesquels l'équivalence est forçée par construction.
k=3 : soit $S_k \in B_k = (i_1,i_2,i_3)$, S_k est obtenue car $\exists S^1_{k-1} \in B_{k-1}$ et $S^2_{k-1} \in B_{k-1}$ telles que $S^1_{k-1} = (i_1,i_2)$ *et* $S^2_{k-1} = (i_2,i_3)$. Mais, si (i_1,i_2) et $(i_2,i_3) \in B_2$ alors (i_1,i_2) et $(i_2,i_3) \in A_2$ (voir k=2) et (comme décrit dans [AS95]) i_2 est une sous-séquence contigüe de (i_1,i_2) et (i_2,i_3) et nous

avons donc $(i_1,i_2,i_3) \in A_3$.

k≥3 :
Pour montrer que $B_k \subseteq A_k \Rightarrow B_{k+1} \subseteq A_{k+1}$ nous utilisons l'hypothèse d'induction suivante : $\forall i \in [0..k]\ B_i \subseteq A_i$.
Soit $S_{k+1} \in B_{k+1} = (i_1,i_2,\ldots,i_{k-1},i_k,i_{k+1})$ nous devons avoir : $\exists S_k \in B_k = (i_1,i_2,\ldots,i_{k-1},i_k)$ et $\exists S_2 \in B_2 = (i_k,i_{k+1})$.
Et d'après l'hypothèse d'induction $S_k \in B_k \Rightarrow S_k \in A_k$
$\Rightarrow (i_2,\ldots,i_{k-1},i_k) \in A_{k-1}$
$\vdots$
$\Rightarrow (i_{k-1},i_k) \in A_2$
De plus nous savons que $(i_k,i_{k+1}) \in A_2$ (car $(i_k,i_{k+1}) \in B_2$) et nous avons donc :
$(i_{k-1},i_k,i_{k+1}) \in A_3$
$\vdots$(Par le processus inverse)
$\Rightarrow (i_2,\ldots,i_{k-1},i_k,i_{k+1}) \in A_k$
donc :

$$\left.\begin{array}{l}(i_1,i_2,\ldots,i_{k-1},i_k) \in A_k \\ (i_2,\ldots,i_{k-1},i_k,i_{k+1}) \in A_k\end{array}\right\} \Rightarrow (i_1,i_2,\ldots,i_{k-1},i_k,i_{k+1}) \in A_{k+1}.$$

4.2 PSP : un algorithme pour l'extraction de motifs séquentiels

La méthode destinée à extraire les séquences maximales de la base de données repose sur le principe générer-élaguer présenté dans [AS95]. L'algorithme général (cf. algorithme 4.2) procède donc par passes successives sur la base de données en alternant génération des candidats puis suppression des candidats non fréquents. Cet algorithme cesse quand plus aucun candidat généré n'est retenu (la longueur maximale des séquences fréquentes dans la base est atteinte). Pour tester chaque séquence de données de la base l'algorithme VERIFYCANDIDATE (cf. algorithme 4.3) utilise une récursion (cf. algorithme 4.4) destinée à atteindre toutes les feuilles possibles en utilisant la séquence de données considérée et en tenant compte des contraintes de temps. Pour cela VERIFYCANDIDATE détermine toutes les tailles et emplacements possibles de la première fenêtre windowSize (i.e. $[l_a,u_a]$) qui sera appliquée à la séquence de données, ensuite l'algorithme FINDSEQUENCE prend en compte cette première fenêtre pour fixer toutes les possibilités d'en créer une seconde ($[l_b,u_b]$). Cette seconde fenêtre sera alors recupérée en entrée de l'appel recursif pour finalement déterminer toutes les combinaisons possibles de windowSize qui seront appliquées à cette séquence de données.

VERIFYCANDIDATE parcourt la séquence d item par item pour tous les couples (l_a,u_a) qu'il parvient à construire. Quand VERIFYCANDIDATE trouve, dans la séquence de données d, un item i dans l'ensemble des items regroupés par (l_a,u_a), il fait un appel à FINDSEQUENCE qui va appliquer ce même comportement à partir de i. En effet, FINDSEQUENCE cherche item par item dans d tout item i' dans l'ensemble des items regroupés par (l_b,u_b) en testant toutes les possibilités de placer (l_b,u_b) et ainsi récursivement jusqu'à atteindre une feuille.

function *generalAlgorithm*
Input : $minGap$, $maxGap$, $windowSize$, un support minimum σ et une base de données DB.
Output : L'ensemble L des séquences fréquentes maximales respectant $minGap$, $maxGap$ $windowSize$ et ayant un pourcentage d'occurrences supérieure à σ.
$k = 1$ // Les itemsets fréquents
$C_1 = \{\{< i >\}/i \in I\}$ $T = C_1$ **while** $C_k \neq 0$ **do**
foreach $d \in DB$ **do** $verifyCandidate(T,d,idSeq(d),k,minGap,maxGap,windowSize)$
$L_k = \{c \in C_k / Support(c) > \sigma\}$ $k = k + 1$ $candidateGeneration(T,k)$ **if** T *is updated by candidateGeneration* **then**
$C_k = T$
endif
$C_k = \emptyset$
endwhile
return $L = \bigcup_{j=0}^{k} L_j$ **end function** *generalAlgorithm*

Algorithme 4.2: *Algorithme général*

function *verifyCandidate*
Input : L'arbre des candidats T. Une séquence de données d et son identifiant $idSeq$. La longueur des candidats à vérifier k. $minGap$, $maxGap$ et windowSize (ws).
Output : L'arbre T vérifiant : $\forall c \in T / c \subseteq d$, $c.compteur$ est incrémenté.
$l_a = firstItemset(d).time()$ **while** $l_a \leq lastItemset(d).time()$ **do**
$u_a = l_a$ **while** $(u_a - l_a) < windowSize$ **do**
$I_p = \{i_p \in d / i_p.time() \in [l_a,u_a]\}$ **foreach** $i_p \in I_p$ **do**
if $i_p \in root.children$ **then**
$depth = 0$ $findSequence(l_a,u_a,root.children(i_p),i_p,d,idSeq,depth)$
endif
endforeach
$u_a = (u_a.succ()).time()$
endwhile
$l_a = (l_a.succ()).time()$
endwhile
end function *verifyCandidate*

Algorithme 4.3: *Algorithme de vérification des candidats*

function *findSequence*
Input : l_a et u_a, deux entiers déterminant la taille de la fenêtre fixée précédement. N un nœud de T, i le dernier item vérifié, la séquence de données d et son identifiant $idSeq$. $depth$ la longueur jusqu'à laquelle les candidats ont été vérifié, $minGap$, $maxGap$ et $windowSize$
Output : L'arbre T vérifiant : $\forall c \in T/prefix(T,N,c)$ *et* $c \subseteq d$, $c.compteur$ est incrémenté.

```
/* Est-ce que N est une feuille? */
if leaf(N) and depth = k then
    if idSeq ≠ N.idLast then
        N.idLast = idSeq  N.cpt++
    endif
else
    /* same transaction */
    I_p = {i_p ∈ d/i_p follows i and i_p.time() ∈ [l_a,u_a]}  foreach i_p ∈ I_p do
        if i_p ∈ N.same then
            findSequence(l_a,u_a,N.same(i_p),i_p,d,idSeq,depth + 1)
        endif
    endforeach
    /* other transaction */
    l_b = (u_a.succ()).time()  /*Contrainte de minGap*/
    while (l_b − u_a) < mingap do l_b = (l_b.succ()).time()  while l_b ≠ lastItem(d).time() do
        u_b = l_b  while (u_b − l_b) < windowSize and (u_b − l_a) < maxgap do
            I_p = {i_p ∈ d/i_p.time() ∈ [l_b,u_b]}  foreach i_p ∈ I_p do
                if i_p ∈ N.other then
                    findSequence(l_b,u_b,N.other(i_p),i_p,d,idSeq,depth + 1)
                endif
            endforeach
            u_b = (u_b.succ()).time()
        endwhile
        l_b = (l_b.succ()).time()
    endwhile
endif
```

Algorithme 4.4: *Algorithme récursif de vérification des candidats*

Client	Date	Items
C1	01/01/1998	1
C1	02/01/1998	2
C1	03/01/1998	3
C1	04/01/1998	4
C2	01/01/1998	1
C2	02/01/1998	2
C2	03/01/1998	3
C2	08/01/1998	4

FIG. 4.7 – *Base de données exemple*

Quand une feuille est atteinte, la séquence candidate évaluée est incluse dans la séquence de données lue et son support est incrémenté.

Exemple 32 *Considérons les clients C_1 et C_2 de la base de données représentée par la figure 4.7.* Pour $l_1 = 1$, l'algorithme PSP est alors conduit à tester les possibilités de combinaisons de séquences vérifiants windowSize illustrées par la figure 4.8. Au total, pour C_1 et en faisant varier l_1 du premier au dernier itemset, PSP va parcourir l'arbre avec pour but d'atteindre toutes les feuilles possibles avec les séquences :

<(1) (2) (3) (4)> <(1) (2) (4)> <(1) (3) (4)>
<(1) (4)> <(1) (2) (3 4)> <(1) (3 4)>
<(1) (2 3) (4)> <(1) (4)> <(1) (2 3 4)>
<(1 2) (3) (4)> <(1 2) (4)> <(1 2) (3 4)>
<(1 2 3) (4)> <(1 2 3 4)> <(2) (3) (4)>
<(2) (4)> <(2) (3 4)> <(2 3) (4)>
<(2 3 4)> <(3) (4)> <(3 4)>
<(4)>

Exemple 33 *La figure 4.9 reprend l'arbre des 3-candidats pour la base de données exemple.* Considérons donc la troisième passe sur la base de données, avec la séquence de données du client C_1 comme entrée pour VERIFYCANDIDATE. PSP peut alors atteindre deux feuilles (car l'arbre représente exactement deux séquences candidates incluses dans la séquence de données) et incrémenter leur support.

4.3 Optimisations

Dans cette section nous revenons à des considérations plus éloignées des contraintes de temps concernant les performances de l'algorithme de recherche de séquences. Nous présentons comment des connaissances acquises sur les items et les séquences de données peuvent être utilisées afin d'éviter des tests coûteux. Enfin nous étudions la façon dont le test d'un candidat peut être accéleré par un sytème de verrous empêchant l'évaluation d'une séquence candidate déjà validée.

séquences testées pour C_1			
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=3	l_4=4, u_4=4
(1)	(2)	(3)	(4)
l_1=1, u_1=1	l_2=3, u_2=3	l_3=4, u_3=4	
(1)	(3)	(4)	
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=3	
(1)	(2)	(4)	
l_1=1, u_1=1	l_2=4, u_2=4		
(1)	(4)		
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=4	
(1)	(2)	(3 4)	
l_1=1, u_1=1	l_2=3, u_2=4		
(1)	(3 4)		
l_1=1, u_1=1	l_2=2, u_2=3	l_3=4, u_3=4	
(1)	(2 3)	(4)	
l_1=1, u_1=1	l_2=4, u_2=4		
(1)	(4)		
l_1=1, u_1=1	l_2=2, u_2=4		
(1)	(2 3 4)		
l_1=1, u_1=2	l_2=3, u_2=3	l_3=4, u_3=4	
(1 2)	(3)	(4)	
l_1=1, u_1=2	l_2=4, u_2=4		
(1 2)	(4)		
l_1=1, u_1=2	l_2=3, u_2=4		
(1 2)	(3 4)		
l_1=1, u_1=3	l_2=4, u_2=4		
(1 2 3)	(4)		
l_1=1, u_1=4			
(1 2 3 4)			

séquences testées pour C_2			
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=3	l_4=4, u_4=4
(1)	(2)	(3)	(4)
l_1=1, u_1=1	l_2=3, u_2=3	l_3=4, u_3=4	
(1)	(3)	(4)	
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=3	
(1)	(2)	(4)	
l_1=1, u_1=1	l_2=4, u_2=4		
(1)	(4)		
l_1=1, u_1=1	l_2=2, u_2=3	l_3=4, u_3=4	
(1)	(2 3)	(4)	
l_1=1, u_1=1	l_2=4, u_2=4		
(1)	(4)		
l_1=1, u_1=2	l_2=3, u_2=3	l_3=4, u_3=4	
(1 2)	(3)	(4)	
l_1=1, u_1=2	l_2=4, u_2=4		
(1 2)	(4)		
l_1=1, u_1=3	l_2=4, u_2=4		
(1 2 3)	(4)		

FIG. 4.8 – *Calculs des combinaisons de windowSize par PSP*

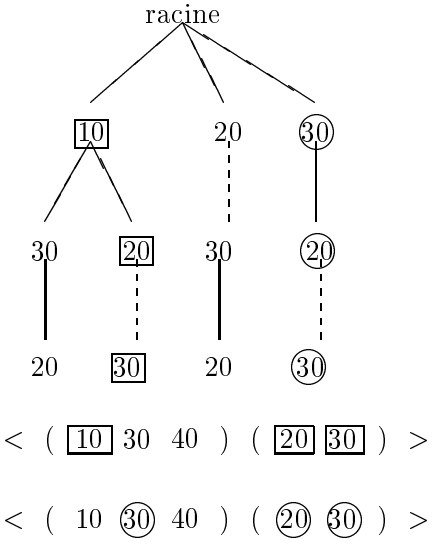

FIG. 4.9 – *Inclusion de candidats dans une séquence de données dans PSP*

4.3.1 Optimisations des passes sur la base de données

Le refus d'items

Dès la première passe sur la base de données effectuée (i.e. après que les items fréquents aient été déterminés), il est possible de procéder à un filtrage des items qu'il sera pertinent de lire dans les passes ultérieures. En effet notre structure présente la propriété suivante : tout nœud fils de la racine contient un item fréquent et tout item fréquent est représenté par un nœud fils de la racine. Les structures utilisées par [AS95, SA96b] ne permettent pas de considérer ce genre de connaissances que le processus de recherche de séquences peut acquérir sur les items au cours de son déroulement. Notre approche, en revanche permet de mettre en place un filtre de la manière suivante : à la lecture d'un item i pour une séquence de données d l'algorithme vérifie si i est fréquent (i.e. si i apparaît dans un fils de la racine) et si ce n'est pas le cas d sera utilisée sans i car il est impossible que i participe à l'incrémentation du support d'une séquence candidate (en effet aucune séquence de données ne contient cet item s'il n'est pas fréquent).

L'inactivation de séquences de données

Au cours des différents parcours de la base de données, il est possible de stocker (par exemple au moyen d'un vecteur de bits) si la séquence de données considérée contribue au support d'une séquence candidate. Il suffit en effet d'observer, après la phase de validation des séquences candidates avec cette séquence de données, si une séquence candidate a vu son support incrémenté par cette séquence de

données. Il est immédiat de constater que si une séquence de données ne contribue à incrémenter le support d'aucune séquence candidate il n'est plus nécessaire de la considérer comme une séquence intéressante pour la suite du processus (en effet chaque séquence candidate au niveau k a pour sous-séquence une séquence candidate du niveau k-1). D'une certaine façon cela revient à réduire, au fil du déroulement du processus, la base de données, pour n'y conserver finalement que les séquences de données qui contiennent une des séquences fréquentes au moins.

Le système de verrous

En vue d'accélérer la validation des séquences de données dans l'arbre des séquences candidates nous avons mis en place un système de verrous permettant de dire pour un nœud n dans l'arbre s'il est nécessaire de valider les séquences ayant pour préfixe la sous-séquence ($racine$, ..., n). Pour cela nous avons doté la structure de données représentant un nœud de l'arbre d'une variable "*checked*" du type booléen et d'une méthode "*getCheck*" dont la définition est donnée dans l'algorithme 4.5. Quand une feuille est atteinte au cours de la vérification à partir d'une séquence de données si son support est supérieur au support demandé alors celle-ci est considérée comme validée et il n'est plus nécessaire de la tester. Lors d'un appel à $getCheck()$, si n est une feuille il fixe sa variable $checked$ à $Vrai$ et fait un appel à $n.father.getCheck()$. Si n est un nœud intermédiaire, il incrémente le nombre de ses fils validés et si ($n.nbValidChildren = n.nbChildren$) alors le nœud n est validé et n fait un appel à $n.father.getCheck()$.
Ainsi, les verrous peuvent remonter dans l'arbre des séquences candidates au fil de la validation des séquences de données. Les séquences de données sont ensuite vérifiées uniquement pour les séquences candidates qui ne sont pas validées. L'exemple 34 illustre l'utilisation de cette méthode.

Exemple 34 *Les trois arborescences représentées à la figure 4.10 illustrent le mécanisme de propagation des verrous.* Considérons l'arbre $\mathcal{A}$: la séquence candidate < (1) (3) (4) > est validée (son support est supérieur au support demandé par l'utilisateur) et le sommet 4 (en caractères gras) de la séquence < (1) (2) (4) > est en cours de validation.
Dans l'arbre $\mathcal{B}$ le sommet 3 de la séquence < (1) (2) (3) > est validé. Au cours de cette validation il fait appel à la méthode $2.getCheck()$. Le sommet 2 présente alors la caractéristique suivante : ($nbValidChildren$=$nbChildren$), il fait donc appel à $1.getCheck()$ (ligne en pointillés entre les sommets 2 et 1). A son tour, le sommet 1 vérifie la condition ($nbValidChildren$=$nbChildren$) et fait appel à $racine.getCheck()$ (ligne en pointillés entre les sommets 1 et racine).
Le résultat de cette propagation est illustré par l'arbre $\mathcal{C}$ où nous pouvons constater que toute une partie de l'arborescence depuis la racine n'est plus considérée pour les prochains tests.

Une autre utilisation de ce mécanisme est faite au début d'une passe sur la base afin de limiter les tests aux seules séquences candidates de la longueur désirée. Pour cela, nous forçons la validation de toute feuille qui n'est pas à une hauteur de k (la longueur des séquences à valider) afin d'éviter les tentatives de validation de cette feuille par le parcours des séquences candidates. La figure 4.11 illustre un arbre dans lequel toutes les feuilles de longueur l ($l \leq k$) ont été validées, ce qui entraîne la validation de sous-arbres grâce à la propagation des verrous.

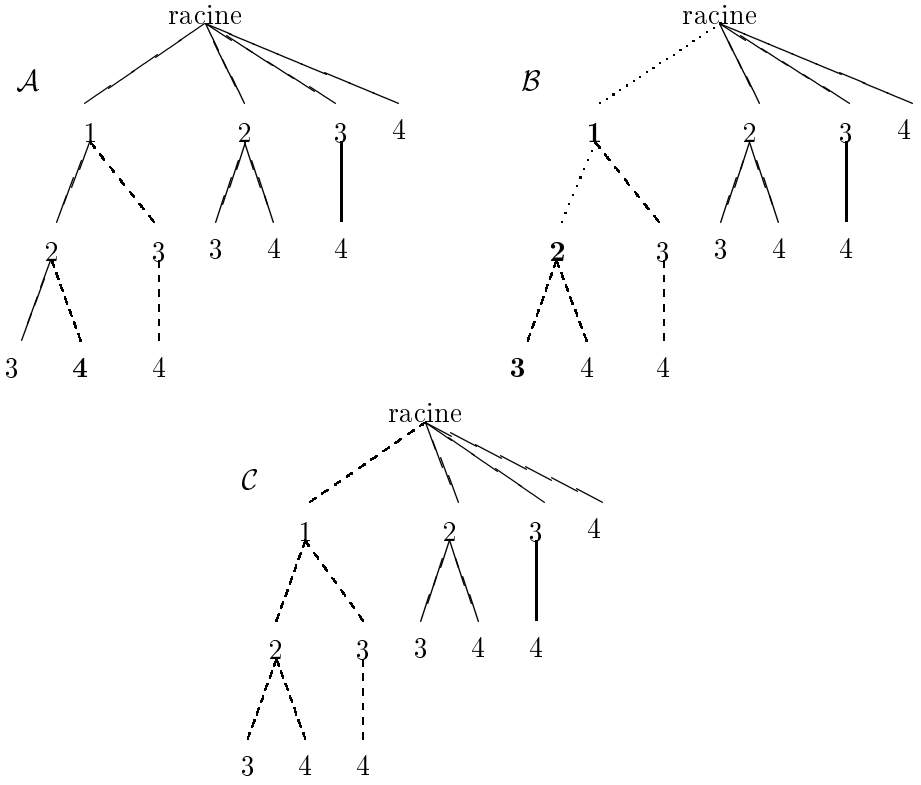

FIG. 4.10 – *Méthode de propagations des verrous*

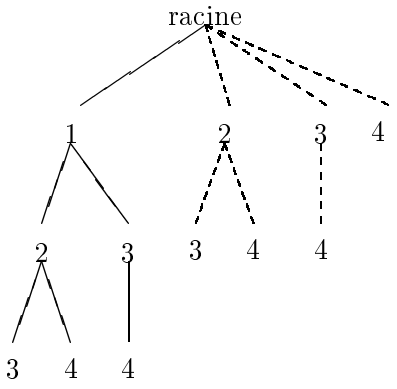

FIG. 4.11 – *Propagations forçée des verrous avant la k^{ieme} passe ($k = 3$)*

procedure *getCheck*
Input : N un nœud de l'arbre des séquences candidates, le support minimal σ
if *feuille(N)* **then**
$N.support++$ **if** $N.support > \sigma$ **then**
$N.father.getCheck()$
endif
else
if $not(racine(N))$ **then**
$lock(requestor)$ $N.nbValidChildren++$ **if** $N.nbValidChildren = N.nbChildren$ **then**
$N.father.getCheck()$
endif
else
$lock(requestor)$
endif
endif
end procedure *getCheck*

Algorithme 4.5: *Propagation des verrous par getCheck*

Conflits entre les deux optimisations : dilemme

L'utilisation des deux dernières optimisations décrites nécessite de faire un choix entre l'une et l'autre. En effet un contre-exemple à la correction des résultats fournis par un processus intégrant les deux techniques est décrit dans l'exemple 35. Il faut alors discuter des différents intérêts que présentent chacune des méthodes et étudier leurs besoins pour décider laquelle sera la plus adaptée.
Nous sommes ici conscients de l'espace mémoire nécessaire pour implémenter une optimisation du type "inactivation de séquences de données". En effet la taille du vecteur destiné à stocker un état pour chaque séquence de données est proportionnel au nombre de séquences de données. Cependant des travaux ont montré qu'il était possible de compacter ce vecteur afin de minimiser ce défaut (cf. [GPW98]). Il est immédiat de constater que le vecteur incriminé contient une majorité de "*Active*" en début de processus et une majorité de "*Inactive*" en fin de processus. Dans un cas comme dans l'autre, il est logique de penser qu'un tel vecteur peut être compacté.
L'espace mémoire peut toutefois rester un obstacle, c'est pourquoi le système de propagation de verrous reste une bonne alternative à l'optimisation des problèmes de recherche de séquences. Cependant il faut reconnaître la priorité que nous accorderions à une optimisation du type "inactivation de séquence de données".

Remarque 2 *: Le choix de la méthode d'optimisation "inactivation de séquence de données" n'empêche cependant pas d'exploiter la technique illustrée par la figure 4.11.*

Exemple 35 *Considérons une base de données DB, deux séquences de données d1 et d2 respectivement lues dans DB et une séquence candidate c1 sous-séquence de d1 et d2.* Admettons que pour k=3, la

séquence de données $d1$ incrémente le support de la séquence candidate $c1$ et qu'à partir de cette opération la séquence candidate $c1$ se trouve validée et donc verrouillée. A la lecture de la séquence de données $d2$ le processus de recherche de séquences ne trouve aucune séquence candidate correspondante. $d2$ ne participe donc pas à l'incrémentation du support d'une séquence candidate et se trouve marquée "Inactive" dans le vecteur des séquences de données actives. Considérons, au cours de la passe suivante (k=4), la séquence candidate $c2$ construite à partir de $c1$. Admettons que $c2$ ne soit pas sous-séquence de $d1$ alors que $c2$ est sous-séquence de $d2$. $c2$ ne verra son support incrémenté ni par $d1$ ni par $d2$ qui est inactive. Les résultats sont alors inexacts.

4.3.2 Optimisations du parcours de l'arbre des candidats

Lors des tests que nous avons effectués avec notre algorithme sur des sources de données comme les fichiers textes ou les schémas (cf chapitre 7), nous avons constaté que PSP présentait des difficultés lors de l'extraction de séquences de taille importante (une discussion sur ce sujet est proposée dans le chapitre 12.2 sous-section 12.2.2). Pour analyser les causes de ces difficultés, mais aussi pour y remédier nous avons mis en place deux optimisations qui permettent d'améliorer les temps de parcours de l'arbre PSP lors de la recherche d'inclusions de candidats dans les séquence de données.

Limitations aux items fréquents

Considérons le cas de figure suivant, pour lequel cinq conditions sont réunies :

- La séquence de données C contient m items
- le nombre d'items fréquents (stockés dans l'arbre) est de f
- les items de C sont répétés jusqu'à n fois (dans C elle même)
- m est *très inférieur* à n.
- m est *très inférieur* à f.

Ce cas de figure est un cas classique dans le contexte du Text Mining. Le problème qui en résulte est un parcours de tout l'arbre avec une séquence qui ne concerne qu'une petite partie de l'arbre. La solution mise en œuvre pour gérer ce cas consiste à scanner la séquence de données afin d'énumérer les items qui la constituent et à se servir de cette information pour verrouiller les parties de l'arbre qui ne seront pas explorées par cette séquence. Cela permet, en d'autres termes, d'obtenir un arbre réduit aux seules branches susceptibles de contenir des candidats inclus dans la séquence de données analysée. Une fois la séquence analysée, les verrous sont supprimés pour pouvoir prendre en compte la même optimisation sur la séquence suivante et pour éviter le problème de conflit entre le système de verrous et celui de l'inactivation de séquence de données, exposé précedemment.

Le verrouillage temporaire

Nous avons remarqué, toujours dans le cadre du Text Mining, qu'un grand nombre de séquences étaient non seulement limitées à quelques items, mais surtout présentaient une très grande taille. Ces séquences sont donc constituées d'une répétition de quelques items un grand nombre de fois, comme par exemple la séquence $s = <(10)\ (20)\ (30)\ (10)\ (20)\ (30)\ (10)\ (20)\ (30)>$. Considérons l'arbre de candidats décrit par la figure 4.12 (candidats de taille 2). Dans cet arbre, le chemin de la séquence $<(10)\ (20)>$,

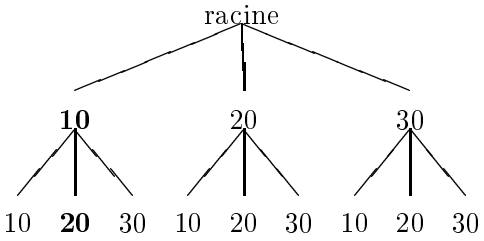

FIG. 4.12 – *optimisation sur la répétition d'items*

<(**10**) (**20**) (30) (10) (20) (30) (10) (20) (30) >
<(**10**) (20) (30) (10) (**20**) (30) (10) (20) (30) >
<(**10**) (20) (30) (10) (20) (30) (10) (**20**) (30) >
<(10) (20) (30) (**10**) (**20**) (30) (10) (20) (30) >
<(10) (20) (30) (**10**) (20) (30) (10) (**20**) (30) >
<(10) (20) (30) (10) (20) (30) (**10**) (**20**) (30) >

FIG. 4.13 – *Les six façons d'inclure* < (10) (20) > *dans s*

représenté en gras, sera parcouru 6 fois par la séquence s (cf. figure 4.13). Or à chaque fois que l'item 20 de ce chemin sera atteint par s, l'identificateur de s, inscrit dans la feuille de valeur 20, empêchera que le compteur de cette feuille soit incrémenté. En d'autre termes, l'item 20 est atteint par s plus d'une fois, ce qui est inutile. Pour éviter de refaire le parcours <(10) (20) >, inscrit en gras dans l'arbre, nous proposons de verrouiller l'item 20 dès que cette feuille sera validée. Le verrou peut alors remonter jusqu'à l'item 10 et ainsi de suite si les frères de 20 sont également verrouillés. Une fois le parcours de l'arbre avec cette séquence terminé, les verrous seront de toutes façons supprimés, en accord avec la première des optimisations présentées.

4.4 Expérimentation

Pour évaluer les performances de l'algorithme et son comportement en fonction de l'augmentation du nombre de transactions, nous avons réalisé des expériences sur une station de travail Enterprise 2 (processeur Ultra Sparc) avec une fréquence d'horloge CPU de 200MHz, 256MB de mémoire centrale et utilisant Unix. Les données sont stockées sur un disque de 9GB Ultra Wide SCSI 3.5".

Pour évaluer les performances de l'algorithme PSP sur une grande quantités de données caractéristiques, nous avons utilisé le générateur de données *synthetic data-gen*[1]proposé par [AS95]. Ce générateur est à l'heure actuelle considéré comme le benchmark typique par la plupart des algorithmes de recherches de règles d'association ou de motifs séquentiels ([AS95, GPW98, SON95, Toi96, BMUT97, SA96b, MTV95]). La génération des données synthétique est expliquée dans [AS95]. Les paramètres utilisés pour la génération de bases de données sont décrits dans la figure 4.14.

1. *http://www.almaden.ibm.com/cs/quest/syndata.html*

$\|D\|$	Nombre de séquence de données
$\|C\|$	Nombre moyen de transactions par séquence de données
$\|S\|$	Longueur moyenne des séquences fréquentes potentielles
N	Nombre d'items

FIG. 4.14 – *Paramètres du générateur de données*

Nom	$\|D\|$	$\|C\|$	$\|S\|$	$\|N\|$
C10-D100-S8-N10	100K	10	8	10000
C20-D1-S10-N1	1000	20	10	1000
C20-D100-S4-N10	100K	20	4	10000

FIG. 4.15 – *Paramètres des fichiers de test*

Trois bases de données différentes ont été utilisées. La figure 4.15 illustre les différents paramètres utilisés par le générateur. La figure 4.16 illustre les résultats obtenus sur les trois bases de données : l'axe des x correspond à une variation du support et l'axe des y représente les temps de réponse. Comme attendu, nous constatons que le coût de l'algorithme dépend fortement de la valeur du support : quand le support diminue, le nombre total de séquences de données augmente et nécessite donc plus de calcul. Pour la base de données C20-D100-S4-N10 la variation de support a été réalisée entre 0.01 et 0.006 dans la mesure où pour un support supérieur, les temps de réponses augmentent linéairement.

Pour vérifier le comportement de PSP lors de l'augmentation du nombre de transactions, nous avons fait varier le nombre de séquences de données $|D|$ de 1 à 100. La figure 4.17 illustre le comportement pour trois valeurs de support : 1%, 1.5% et 2%. Nous remarquons que PSP affiche un comportement linéraire avec des temps de réponse proportionnels à l'augmentation du nombre de séquences, quel que soit le support spécifié.

Une comparaison des résultats des algorithmes PSP et GSP est donnée dans la section 4.5.

Nous avons également utilisé un fichier Access_log qui contient toutes les connections à un serveur http. La structure du fichier est la suivante : identification de la machine et de l'utilisateur, date de la connection, opération effectuée, message d'erreur et nombre de bits envoyés. La section 8.1 décrit en détail les différents paramètres ainsi que le traitement nécessaire pour pouvoir manipuler un tel fichier. Le fichier utilisé a une taille de 10MB et concerne 2000 clients (machines) et 1500 items (opérations effectuées sur le serveur). La figure 4.18 illustre les résultats de l'expérimentation.

4.5 Discussion

L'algorithme GSP ne peut pas faire preuve d'une grande efficacité à cause de la nature même de la structure qu'il met en œuvre. En effet l'approche présentée dans ce chapitre montre qu'une structure efficace permet de retrouver plus rapidement une séquence dans un ensemble de séquences candidates.

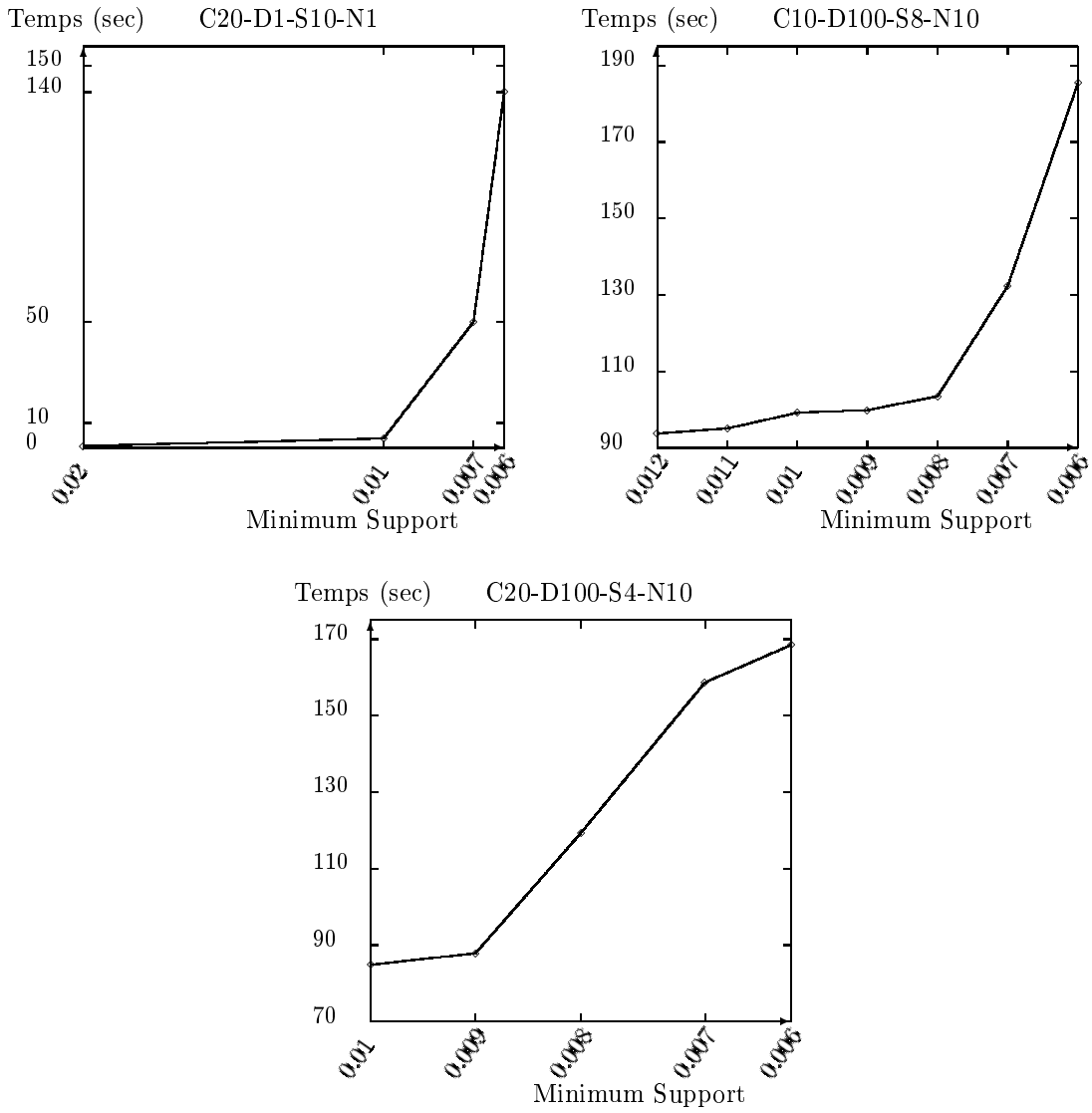

FIG. 4.16 – *Temps d'exécution de PSP*

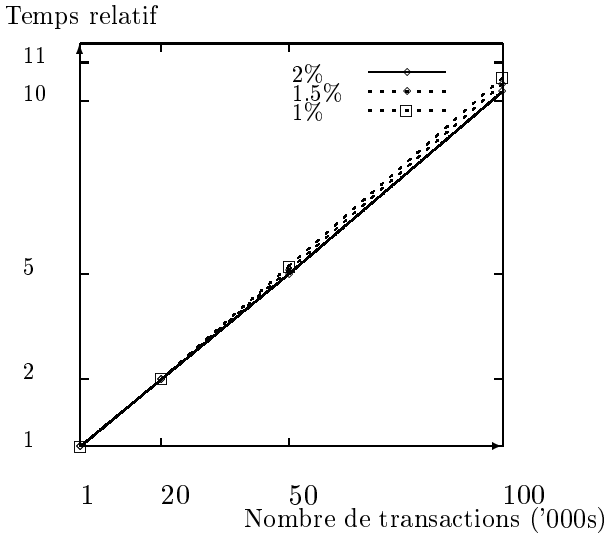

FIG. 4.17 – *Linéarité comportementale de PSP*

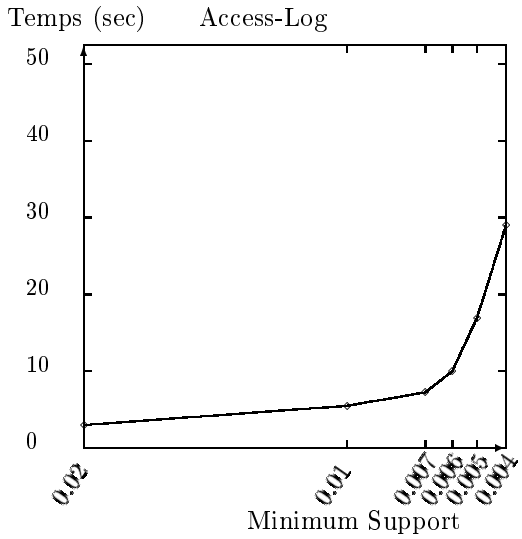

FIG. 4.18 – *Temps d'exécution de PSP sur un fichier Access-Log*

Pour mieux illustrer l'avantage de cette structure, considérons l'exemple suivant :

Exemple 36 *La figure 4.19 illustre les structures utilisées par notre approche (arbre de gauche) et par l'approche GSP (arbre de droite).* Les 2-sequences de l'exemple 27, permettent d'obtenir C_3, l'ensemble des 3-candidats suivants : < (10) (10 40) > < (10) (30) (20) > < (10) (30 40) > < (30 40 10) > < (40) (10 30) > < (40) (1040) >. Nous pouvons constater que la proposition 1 est vérifiée et que notre structure nécessite moins d'espace mémoire que celle de GSP.

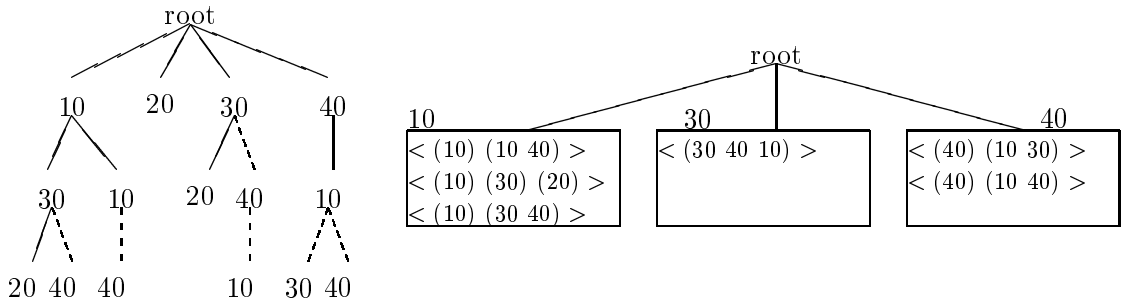

FIG. 4.19 – *Les deux structures mises en jeu.*

Pour évaluer quels candidats sont inclus dans une séquence de données, GSP parcourt l'arborescence jusqu'à atteindre une feuille puis applique un algorithme mettant en œuvre un backtrack coûteux sur toutes les séquences candidates contenues dans cette feuille en parcourant chaque séquence candidate dans son intégralité. Notre approche permet de se limiter à la recherche des feuilles sans plus de calcul, quand une feuille est atteinte la seule opération restante est d'incrémenter son support.
Les séquences contenues dans les feuilles de la structure de [AS95] sont groupées dans des feuilles quand elles ont un prefixe commun mais GSP ne profite pas de cette organisation car il teste chaque séquence candidate de la feuille du premier au dernier item. Notre approche utilise le fait que toutes les feuilles "sœurs" ont un préfixe commun qu'il n'est pas nécessaire de tester pour chacune d'entre elles. Cela peut même se généraliser à l'argument suivant : quand le nœud N est atteint, le chemin parcouru de la racine vers ce nœud n'est plus à re-parcourir pour atteindre son frère. C'est là, bien sûr, une propriété commune à tout parcours d'arbre élémentaire, cependant c'est une propriété dont nous pouvons profiter grâce à une structure qui distingue les changements d'itemsets dans une séquence candidate offrant ainsi la possibilité de toujours stocker les séquences candidates voulues.

Nous avons également implémenté une version approchée de GSP (GSP-*like*) offrant la possibilité de comparer les deux approches. La structure utilisée par GSP-*like* est un ensemble de séquences, calculé à partir de l'arbre des candidats et utilisé pour vérifier tous les candidats par rapport aux séquences de données. Il s'agit en fait du pire des cas de GSP (celui où chaque feuille peut être atteinte depuis la racine quelle que soit la séquence de données) car nous considérons alors que l'arbre est réduit à une seule feuille contenant toutes les séquences. Il est évident que cette comparaison présente des limites dans la mesure où la structure utilisée par GSP diffère sur le nombre de feuilles. Cependant, il faut considérer le fait que l'efficacité de GSP dépend de la taille des feuilles et GSP-*like* n'est autre qu'un GSP utilisant des feuilles de grande taille. De plus, si nous considérons le cas opposé, GSP utiliserait

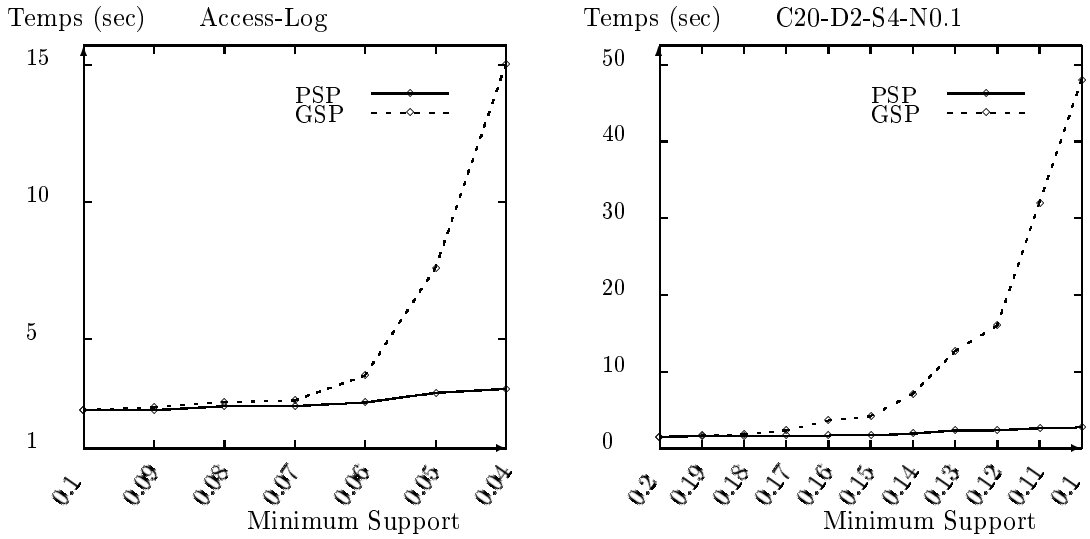

FIG. 4.20 – *Temps d'exécution comparés de PSP et GSP*

alors une structure arborescente dont les feuilles ne contiendrait qu'une ou deux séquences. Dans un tel cas de figure, il faut alors prendre en considération que pour chacune des feuilles atteintes, GSP lance, à nouveau, un test sur chacune des séquences contenues dans la feuille (en effet, GSP n'a aucun autre moyen de savoir si la séquence candidate atteinte est incluse dans la séquence de données considérée). Enfin, pour conclure cette comparaison, admettons que dans le cas précédemment cité, GSP écarte de l'ensemble des solutions un ensemble de séquences candidates réparties dans un ensemble de feuilles ayant pour prédécesseur commun un nœud N, GSP évite alors d'effectuer des calculs sur ces séquences. De la même manière PSP ne trouvant aucune possibilité de dépasser le nœud N ne sera pas conduit à effectuer le moindre calcul pour les successeurs de N. Ainsi nous pouvons affirmer que GSP-*like* est un outil de comparaison significatif et fiable car il nous permet de connaître le comportement de GSP dans des cas défavorables, sachant que dans les cas favorables, GSP ne tire aucun avantage de sa structure par rapport à PSP. La figure 4.20 représente les séries de tests comparatifs destinés à mesurer les écarts de performance entre GSP et PSP. La courbe de gauche représente les temps de réponse pour un fichier Access-Log et la courbe de droite les temps de réponse comparés pour un fichier de données généré par synthetic-datagen.

Chapitre 5

Motifs séquentiels généralisés

Dans le Chapitre 4 nous avons présenté un algorithme et une structure de données destinés à résoudre efficacement le problème de la recherche de séquences maximales. Si la structure semble bien adaptée au problème il semble, en revanche, que l'algorithme mis en œuvre n'exploite pas toutes ses possibilités. En effet l'algorithme PSP peut se trouver mis en défaut dans certaines configurations. Afin d'illustrer ce problème considérons la séquence de données suivante : < (1) (2) (3) (4) (5) > extraite de la base représentée par la figure 5 :

Client	Date	Item
C1	01/04/98	1
C1	03/04/98	2
C1	04/04/98	3
C1	07/04/98	4
C1	17/04/98	5

FIG. 5.1 – *Exemple de base de données*

Avec windowSize=1, minGap=1 et maxGap=19, l'algorithme teste les combinaisons suivantes afin de vérifier si elles sont susceptibles d'incrémenter le support d'une séquence candidate :

Sans contraintes de temps < (1) >
⋮
< (5) >
⋮< (1) (2) >
⋮
< (4) (5) >
⋮
< (1) (2) (3) (4) (5) > puis en tenant compte de windowSize et de minGap :
< (2) (5) > •
⋮
< (1) (3) (5) > •
⋮
< (1) (2 3) (5) > •
⋮
< (1) (2) (4) (5) > •
< (1) (3) (4) (5) > •
< (1) (2 3) (4) (5) > ∘

Nous remarquons alors que les séquences marquées par un • sont incluses dans la séquence marquée

par un $\circ$ et que les séquences marquées par un $\cdot$ sont incluses dans la séquence marquée par un $\odot$. En effet si une séquence candidate fait partie du support pour $<$ (1) (2 3) (4) (5) $>$ alors elle fait également partie du support pour $<$ (1) (2) (4) (5) $>$ et $<$ (1) (3) (4) (5) $>$, ce qui rend le test de ces deux séquences inutile car elles sont incluses dans une autre, plus générale et de plus cela contredirait la remarque 1 (page 51).

La solution à ce problème, décrite dans la suite de ce chapitre, est de pré-calculer, grâce à l'algorithme GTC (Graph for Time Constraints), un ensemble pertinent de séquences à tester pour une séquence de données. En pré-calculant cet ensemble, nous nous fixons deux buts qui permettent de réduire le temps nécessaire à l'analyse d'une séquence de données par rapport à l'arbre des séquences candidates :

- le parcours de l'arbre des séquences candidates doit être indépendant des contraintes de temps définies par l'utilisateur. Cela implique un parcours sans "backtrack" permettant tout de même de vérifier toutes les combinaisons de windowSize possibles qui sont, pour l'instant, calculées "à la volée" ;
- ce parcours doit vérifier le moins de combinaisons possibles, surtout si un certain nombre de ces combinaisons suffit à tester l'ensemble des candidats nécessaire, c'est à dire un ensemble de combinaisons qui regroupe toutes celles possibles (par exemple $<$ (1) (2 3) (4) (5) $>$, $<$ (1) (2) (3) (4) (5) $>$ dans l'exemple précédent).

Nous étudions dans la section 5.1 l'algorithme GTC et la structure de données qu'il utilise, qui sont destinés à éviter de calculer les combinaisons de windowSize pendant le parcours de l'arbre et ne vérifier que ce qui est pertinent.

La sous-section 5.1.1 décrit une première version de GTC, destinée à présenter une nouvelle approche du problème de la représentation des séquences de données et de la vérification des séquences candidates. Il s'agit de proposer une solution au parcours de l'arbre, en ne considérant dans un premier temps que la contrainte de minGap. Nous y voyons comment le parcours de l'arbre des séquences candidates est adapté à cette nouvelle approche pour satisfaire un premier but : éviter de tester des séquences incluses et ne pas calculer les possibilités de minGap pendant le parcours. La sous-section 5.1.2 décrit la façon dont l'algorithme a évolué pour résoudre également le problème de windowSize tout en gardant les avantages de la première version de GTC (il s'agit de l'algorithme GTC_{ws}). Enfin, la version finale de GTC, présentée dans la section 5.2, expose la solution proposée aux problèmes des contraintes de temps, avec la prise en compte de maxGap. Des optimisations sont expliquées dans la section 5.3 et les expérimentations, destinées à évaluer les performances de notre algoritmes sont décrites à la sections 5.4. Nous montrons également que cette approche permet de fournir les mêmes résultats que l'approche vue dans le Chapitre 4 et une discussion, proposée par la section 5.5, conclut ce chapitre.

5.1 GTC : le graphe solution aux contraintes de temps

5.1.1 Prise en compte de minGap

Nous commençons par donner une présentation intuitive de ce qui va suivre et du comportement de GTC, l'algorithme destiné à résoudre les contraintes de temps appliquées au problème de la recherche

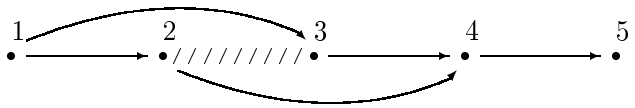

FIG. 5.2 – *Visualisation d'une séquence de données*

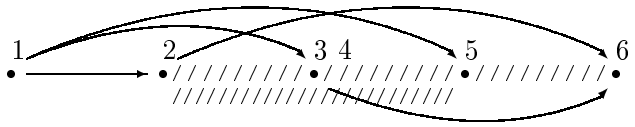

FIG. 5.3 – *Un minGap plus contraignant*

de motifs séquentiels. Cette présentation sera suivie par une définition plus formelle et détaillée.

Reprenons le problème de la contrainte de sans tenir compte, pour l'instant, de maxGap ni de windowSize. L'idée directrice qui nous intéresse ici consiste à dire que le parcours de l'arbre des séquences candidates doit être simplifié. Pour cela nous devons trouver un outil permettant de calculer à l'avance les combinaisons de séquences issues de la prise en compte des contraintes de temps sur une séquence de données.
Considérons la représentation graphique d'une séquence de données (cf. figure 5.2) correspondant à la base de l'exemple 5. Le séparateur //////// entre deux items a et b (ex : (2, 3 4) ou (2, 5) de la figure 5.3) indique que a et b sont trop proches au sens de la contrainte de minGap. Nnous pouvons extraire deux séquences qu'il est utile de chercher dans l'arbre des séquences candidates : < (1) (2) (4) (5) > et < (1) (3) (4) (5) >, qui correspondent aux deux chemins de source "1" du graphe.

Problème des chemins de taille maximale respectant minGap

Définition 9 Une *sous-séquence de données* est une séquence de données respectant la contrainte maxGap.

Définition 10 Un *graphe de séquence* G(S,A), où S est l'ensemble des sommets et A l'ensemble des arcs du graphe, est un graphe orienté acyclique représentant une sous-séquence de données. G est tel que les sommets sont les items de la séquence de données tout chemin ayant un sommet source pour origine et un sommet puits pour arrivée n'est inclus dans aucun autre et toute séquence candidate supportée par la séquence de données représentée est incluse dans un tel chemin au moins.

Définition 11 Un *chemin de séquence* est un chemin du graphe de séquence ayant pour sommet de départ un sommet source du graphe de séquence et pour sommet d'arrivée un sommet puits du graphe de séquence. L'ensemble SP des chemins de séquence représente les séquences à tester dans l'arbre des

séquences candidates : pour une séquence de données d, quelle que soit c une séquence candidate telle que c est sous-séquence de d, il existe p appartenant à SP, l'ensemble des chemins de séquence calculé à partir du graphe de séquence de d, tel que p supporte c.

Exemple 37 *Considérons la base de données représentée par la figure 5.4.* Pour un minGap de 2, la figure 5.3 représente le graphes de séquences correspondant, déduit de la séquence de données du client $C1$. La figure 5.2 représente les des deux sous-séquences de données correspondants à la base de données de la figure 5.

Client	Date	Item
C1	01/04/98	1
C1	05/04/98	2
C1	06/04/98	3 4
C1	07/04/98	5
C1	09/04/98	6

FIG. 5.4 – *Exemple de base de données*

Nous sommes donc conduits à résoudre le "Problème des chemins de taille maximale respectant minGap" qui consiste à enumérer, pour une sous-séquence de données s, l'ensemble SP des chemins de séquence à tester dans l'arbre des séquences candidates A et vérifiant les deux propriétés suivantes :

1. $\forall s1,s2 \in SP/s1 \not\subset s2$.
2. $\forall c \in A/s$ *supporte* c, $\exists p \in SP/p$ *supporte* c.
3. $\forall p \in SP, \forall c \in A/p$ *supporte* c, *alors* d *supporte* c.

Ce problème est résolu par l'algorithme GTC qui fournit une solution optimale (Théorème 3) en $O(n^2)$ (Théorème 4).

Construction du graphe de séquence

Pour trouver l'ensemble complet des chemins minimaux, GTC utilise la structure de données suivante : un graphe orienté classique dont chaque sommet représente un itemset et possède un vecteur *isPrev* de taille $|S|$ indiquant pour ce sommet quel autre sommet le précède par un chemin quelconque (il s'agit donc d'un vecteur de booléens qui indique au sommet x, pour chaque sommet y si y est prédécesseur de x ou non). L'algorithme de construction du graphe explore chaque sommet x pour lui appliquer une méthode qui peut se décomposer en deux étapes :

1. **Phase d'exploration** : l'algorithme cherche le premier sommet y qui vérifie $(y.date() - x.date() > minGap)$ et construit l'arc (x,y). Ensuite pour chaque sommet z tel que $(z.date() - y.date() > minGap)$ l'algorithme met à jour $z.isPrec[x]$ pour conserver le fait que x atteindra z quoiqu'il arrive en passant par y. Soit Z l'ensemble des sommets z ainsi traités.
2. **Phase de complétion** : le but dans cette phase est d'utiliser l'information "x atteindra $z (z \in Z)$" pour construire les arcs (x,t) sachant que $(t.date() - x.date() > minGap)$ et $t.isPrec[x] \neq 1$ (donc $t \notin Z$ et x n'atteindra pas t si l'on ne construit pas l'arc (x,t)).

function GTC_{min} **return** graphe de séquence
Input : une sous-séquence de données d
Output : G(S,A) le graphe de séquence de d
foreach *itemset* $i \in d$ **do**
$S = S \cup \{i\}$;
endforeach
foreach $x \in S$ **do**
//Phase 1
$y = x$;
while $y.begin() - x.begin() < minGap$ **do** $y++$;
$A = A \cup \{x,y\}$;
$i_p = \{i \in S / i.begin() - y.begin() > minGap\}$;
foreach $z \in i_p$ **do**
$z.isPrec[x] = 1$;
endforeach
// Phase 2
$j_p = \{j \in S / j.begin() > y.begin()\ and\ j.isPrec[x] = 0\}$;
foreach $t \in j_p$ **do**
$A = A \cup \{x,t\}$;
endforeach
endforeach
end function GTC_{min}

Algorithme 5.1: *Pré-version de GTC, solution pour minGap*

Une fois l'algorithme GTC_{min} appliqué à une sous-séquence de données, l'ensemble SG des séquences à tester dans l'arbre des séquences candidates se construit en cherchant dans le graphe tous les chemins de séquence.

Exemple 38 *La figure 5.5 représente quatre étapes de l'algorithme de construction du graphe.*

- $x = 1$: Première phase. l'algorithme cherche le premier sommet y tel que $y.date() - x.date() > minGap$, il s'agit du sommet 2. Ensuite les sommets 4 et 5 étant accessibles depuis le sommet 2 (i.e. les arcs (2,4) et (2,5) respectent la contrainte de minGap), leurs composants $isPrec$ est mis à jour ($4.isPrec[1] = 1$ et $5.isPrec[1] = 1$) pour indiquer que ces sommets sont accessibles à partir de 1 mais par l'intermédiaire d'un chemin plus long que l'arc (1,4) ou (1,5).
  Deuxième phase : pour tous les sommets t successeurs de 2 qui vérifient $minGap(x,t)$ et tels que $t.isPrec[1] = 0$, l'algorithme construit l'arc (x,t) (ici seul le sommet 3 repond aux deux critères).
- $x = 2$: Première phase. 4 est le premier sommet accessible depuis 2 donc l'arc (2,4) est construit. 5 est le seul successeur de 4 donc $5.isPrec[2] = 1$. Deuxième phase : il n'y a aucun sommet t tel que $minGap(2,t)$ et $t.isPrec[2] = 0$ soit vérifié.
- $x = 3$: Première phase. 4 est le premier sommet accessible depuis 3 donc l'arc (3,4) est construit. 5 est le seul successeur de 4 donc $5.isPrec[3] = 1$. Deuxième phase : il n'y a aucun sommet t tel que $minGap(3,t)$ et $t.isPrec[3] = 0$ soit vérifié.

– $x = 4$: Première phase. Nous construisons l'arc (4,5). La deuxième phase est annulée car 5 n'a pas de successeur (au sens de la datation).

Exemple 39 *La figure 5.6 illustre deux étapes de GTC.*

– $x = 1$: Première phase. l'algorithme cherche le premier sommet y tel que $y.date() - x.date() > minGap$, il s'agit du sommet 3. Ensuite le sommet 5 étant accessible (au sens où minGap ne l'empêche pas) depuis le sommet 3, il est modifié par $5.isPrec[1] = 1$.
Deuxième phase : pour tous les sommets t successeurs de 3 qui vérifient $minGap(x,t)$ et tels que $t.isPrec[1] = 0$, l'algorithme construit l'arc $(1,t)$ (ici seul le sommet 4 repond aux deux critères).
– $x = 2$: Première phase. 4 est le premier sommet accessible depuis 2 donc l'arc (2,4) est construit. Il n'y a aucun successeur de 4 qui vérifie minGap. Deuxième phase : seul 5 vérifie les deux critères $5.isPrec[2] = 0$ et $minGap(2,5)$ donc l'arc (2,5) est construit.
– $x = 3$: Première phase. 5 est le premier sommet accessible depuis 3 donc l'arc (3,5) est construit. La deuxième phase est annulée car 5 n'a pas de successeur (au sens de la datation).
– $x = 4$: Il n'y a ni première phase ni deuxième phase car il y a un minGap entre 4 et 5 et de plus 5 est le dernier sommet du graphe.

Théorème 3 *L'algorithme GTC_{min} construit exactement toutes les solutions de la plus grande taille possible pour le problème des chemins respectant minGap.*

Démonstration : Nous prouvons, dans un premier temps, que l'inclusion de chemins de séquence est impossible après l'exécution de GTC_{min}. Ensuite, nous montrons que pour toute séquence candidate c supportée par une séquence de données s, il existe un chemin de séquence dans le graphe de séquence calculé à partir de s, qui représente une séquence support pour c.
supposons qu'il existe $s1$ et $s2$, deux chemins de séquence dans SG, l'ensemble des séquences à tester, tels que $s1 \subset s2$. Cela voudrait dire que le graphe de séquence contient le sous-graphe représenté par la figure 5.7, c'est à dire qu'il existerait un chemin $(a,\ldots,c)$ de longueur ≥ 2 et un arc (a,c). Or, si le chemin (a,c) existe, nous avons $c.isPrec[a] = 1$. Il y a en effet deux possibilités d'avoir un chemin de longueur ≥ 1 de a vers b : un arc (a,b) ou un chemin $(a,\ldots,b)$. Dans le premier cas, $c.isPrec$ est immédiatement mis à jour par l'instruction $c.isPrec[a] \leftarrow 1$ et dans le second cas il existe a' un sommet du chemin $(a,\ldots,b)$ tel que l'arc (a,a') fait partie de ce chemin, auquel cas l'instruction $c.isPrec[a] \leftarrow 1$ a eu lieu lors de la construction de l'arc (a,a'). Donc, après la construction du chemin $(a,\ldots,b,\ldots,c)$ nous avons bien $c.isPrec[a] = 1$ et la construction de l'arc (a,c) est rendue impossible. Le sous-graphe représenté par la figure 5.7 est donc impossible après l'exécution de GTC_{min}.
Pour montrer que pour toute séquence candidate c supportée par une séquence de données s, il existe un chemin dans le graphe de séquence calculé à partir de s qui représente une séquence support pour c, nous allons exploiter le fait que GTC_{min} explore tous les sommets du graphe et ne construit pas de séquences incluses. L'algorithme parcourt tous les sommets du graphe, ce qui implique que si un chemin respectant minGap contient le sommet x, alors ce chemin fait partie du graphe après l'exécution de GTC_{min} (même si il est restreint au seul sommet x). Tous les sommets font partie d'un chemin, l'inclusion de chemin est impossible et si deux chemins $(x,\ldots,y)(y',\ldots,z)$ peuvent être fusionnés (i.e. $y'.date() - y.date() > minGap$), ils le seront car l'algorithme en explorant le sommet y va construire l'arc (y,y') (ce raisonnement peut s'étendre à "si deux chemins $(x,\ldots,y)(y',\ldots,z)$ peuvent être fusionnés

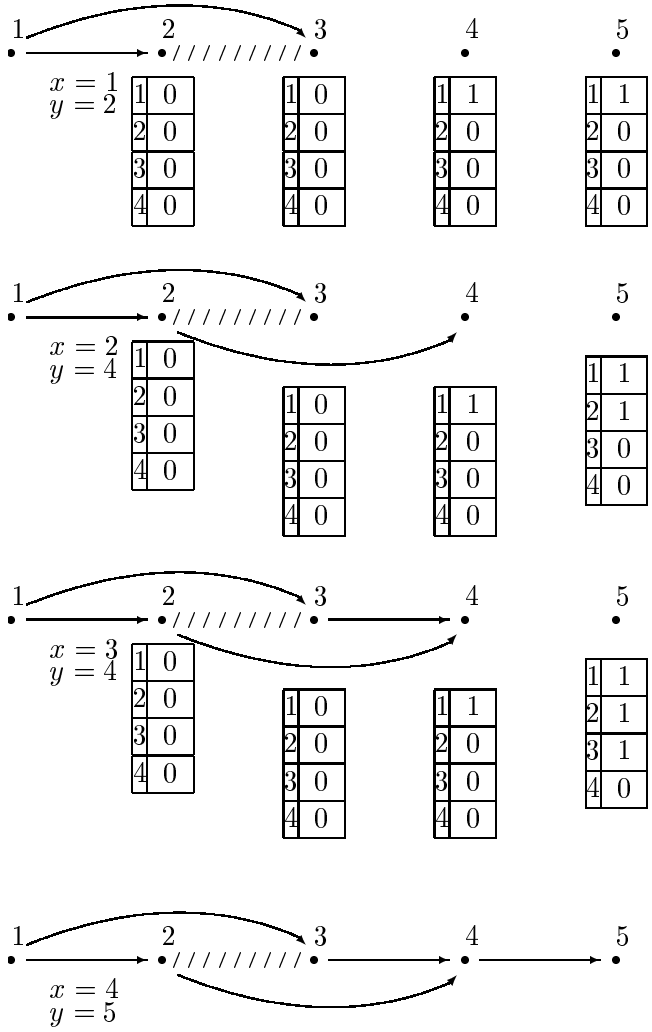

FIG. 5.5 – *Quatre étapes de GTC*

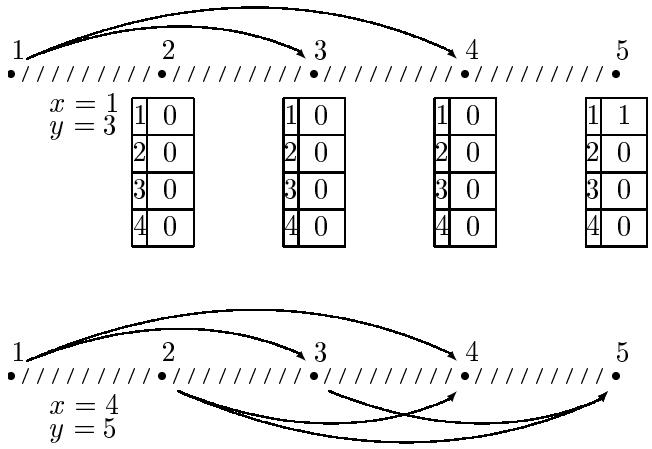

FIG. 5.6 – *Deux étapes de GTC*

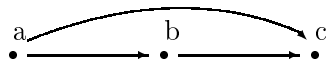

FIG. 5.7 – *Schéma d'inclusion minimum*

(i.e. $y'.date() - y.date() > minGap$) ils le seront car l'algorithme après avoir exploré le sommet y va construire le chemin $(y,\ldots,y')$"). L'algorithme GTC$_{min}$ construit donc exactement toutes les solutions de la plus grande taille possible pour le problème des chemins respectant minGap.□

Théorème 4 *L'algorithme GTC$_{min}$ a une complexité en O(n^2).*

Analyse en complexité

Le pire des cas se présente lorsque minGap=0. Il est inutile d'utiliser GTC$_{min}$ dans un tel cas de figure, dans la mesure où PSP est capable de résoudre le problème aussi bien sinon mieux que GTC$_{min}$ (cf. sections 5.5 et 5.4). Pourtant nous analysons la complexité de GTC$_{min}$ malgré cet aspect car il sera indispensable lors de son évolution vers la prise en compte de windowSize.
Si minGap=0, GTC$_{min}$ parcours le graphe sommet par sommet et pour chaque sommet x, évalue la distance entre x et chacun de ses successeurs y. Si minGap=0 alors le premier successeur y de x rencontré est pris en compte. A partir de y, GTC$_{min}$ considère tous les successeurs de y à deux reprises (une fois pour y, i.e. la phase de propagation et une fois pour x, i.e. la phase de gapJumping).
Le nombre d'opérations nécessaire peut alors s'exprimer sous la forme d'une suite arithmétique : $\sum_{k=1}^{n-1} 2k - 1$.

5.1.2 Prise en compte de minGap et windowSize

Le nombre de séquences incluses qui pose problème, comme nous l'avons vu au début de ce chapitre, est encore plus important si nous considérons la contrainte de windowSize. En effet avec la base de données représentée par la figure 5.8, ws=5 et mingap=1, l'algorithme PSP de la première approche teste les séquences suivantes (nous ne considérons ici que les séquences calculées avec windowSize, le problème de minGap ayant été mis en évidence dans l'introduction de ce chapitre) :

< (1) (2) (3) (5) (6) > •
< (1) (2) (3) (4) (6) > •
< (1) (2) (3) (4 5) (6) > ∘
< (1) (2) (3 4) (6) > ·
< (1) (2) (3 4 5) (6) > ⊙

Comme l'indiquent les marques apposées aux côtés des séquences, seulement deux d'entres elles sont nécessaires car elles couvrent l'ensemble de ce que teste PSP :

< (1) (2) (3) (4 5) (6) >
< (1) (2) (3 4 5) (6) >

Comme nous l'avons, l'exploration de l'arbre pour découvrir les séquences candidates supportées par la séquence de données est un processus très coûteux qui, répété un trop grand nombre de fois, peut faire "s'effondrer" un algorithme de recherche de séquences. Nous exploitons cette connaissance concernant l'inclusion des séquences testées afin de limiter le nombre de parcours de l'arbre et donc optimiser les temps de réponse.
Pour prendre en compte la contrainte windowSize nous étendons l'algorithme GTC_{min} en calculant les combinaisons cohérentes de windowSize en début d'algorithme puis, une fois le graphe respectant minGap calculé, l'algorithme détecte alors les inclusions. Le résultat de cette manipulation est illustré par la figure 5.9 qui représente le graphe de séquence de la base représentée par la figure 5.8.

Client	Date	Item
C1	01/04/98	1
C1	07/04/98	2
C1	13/04/98	3
C1	17/04/98	4
C1	18/04/98	5
C1	24/04/98	6

FIG. 5.8 – *Exemple de base de données*

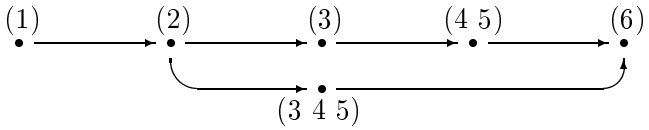

FIG. 5.9 – *Un graphe de séquence calculé avec windowSize*

Construction du graphe de séquence

Pour énumerer les combinaisons de windowSize possibles, l'algorithme parcours chaque sommet x, et détermine pour chacun d'entre eux quels autres sommets peuvent être "fusionnés" avec x (x et y peuvent fusionnés si l'on peut grouper les itemsets qu'ils représentent, i.e. si $y.date() - x.date() < windowSize$). Nous adaptons la structure de données utilisée pour prendre en considération cette fusion. Désormais chaque sommet représente un itemset et possède un vecteur *isPrev* de taille $|S|$ indiquant pour ce sommet quel autre sommet le précède par un chemin quelconque et un itemset possède une date de début et une date de fin qui seront accédés par $i.begin()$ et $i.end()$.

Définition 12 *Un itemset i est inclus dans un autre itemset j si et seulement si les deux conditions suivantes sont respectées :*

- $i.begin() \geq j.begin()$,
- $i.end() \leq j.end()$.

Une fois le graphe respectant minGap calculé (cf. algorithme GTC_{min} section 5.1.1), l'algorithme détecte les inclusions de la manière suivante : pour chaque sommet x, déterminer l'ensemble de ses successeurs : $x.next$. Pour chaque sommet y dans $x.next$, si $y \subset z$,$z \in x.next$ et $y.next \subseteq z.next$ alors retirer le sommet y du graphe.

procedure *addWindowSize*
Input : l'ensemble des sommets S à traiter (trié par ordre croissant (1^{ere} clé : date de début, 2^{nde} clé : date de fin)).
$a = S.first()$ **while** $a \neq S.last()$ **do**
  $b = S.succ(a)$ **while** $b.end() - a.begin() < windowSize$ **do**
    $i = group(a,b)$ //insère le sommet i avant le sommet b dans S
    $S.insert(i,b)$ $a = S.succ(a)$ $b = S.succ(a)$
  endwhile
  $a = S.succ(a)$
endwhile
end procedure *addWindowSize*

Algorithme 5.2: *Calcul des combinaisons possibles pour windowSize*

Exemple 40 *Considérons la base de données représentée par la figure 5.8,* le graphe résultant de la première étape est représenté par la figure 5.10. En effet, windowSize étant fixé à 4, les items 3, 4 et 5

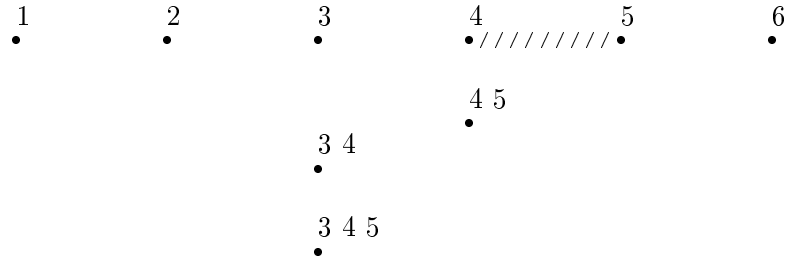

FIG. 5.10 – *graphe de séquence après la première étape*

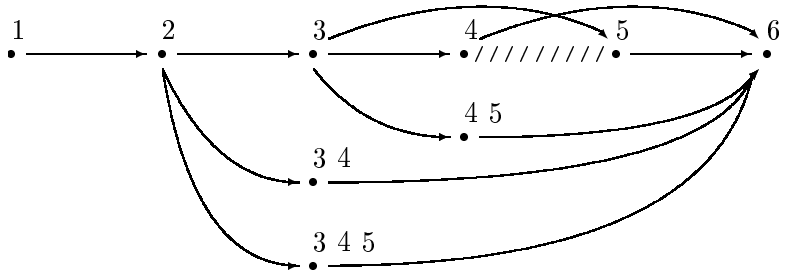

FIG. 5.11 – *GTC avec windowSize après la deuxième étape*

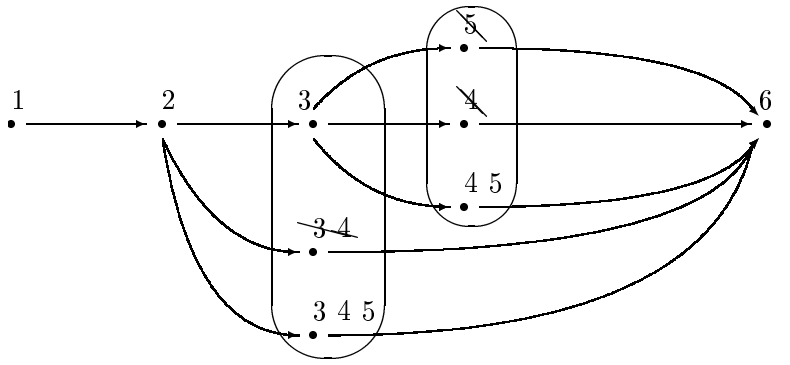

FIG. 5.12 – *Méthode de détection d'inclusions*

procedure *pruneIncluded*
Input : le graphe $G(S,A)$ dans lequel il faut traiter les cas d'inclusions dûs à windowSize
foreach $x \in S$ **do**
  foreach $y \in x.next$ **do**
    foreach $z \in x.next$ **do**
      if $y \subset z$ *and* $y.next \subseteq z.next$ **then** $prune(y)$
    endforeach
  endforeach
endforeach
end procedure *pruneIncluded*

Algorithme 5.3: *Détection et suppression des inclusions dues à windowSize*

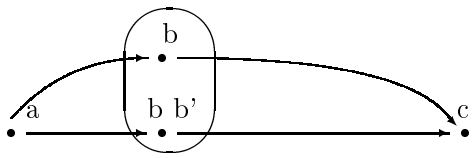

FIG. 5.13 – *Schéma d'inclusion minimum dû à windowSize*

peuvent être fusionnés. Cependant, il ne faut pas négliger la possibilité que < (3) (4 5) > ou encore < (3 4) (5) > peuvent faire partie d'une séquence candidate et donc doivent être testées. C'est pourquoi l'algorithme construit les sommets qui représentent les itemsets (3 4), (4 5) et (3 4 5).
le graphe résultant de la seconde étape est illustré par la figure 5.11. Il s'agit du graphe après l'exécution de la première version de GTC. L'adaptation pour prendre en compte les possibilités ajoutées par windowSize se fait donc de manière immédiate.
La méthode utilisée pour détecter l'inclusion est illustrée figure 5.12. Nous constatons que les séquences < (3) (4) (6) > et < (3) (5) (6) > sont incluses dans la séquence < (3) (4 5) (6) >. En effet $3.next$={(4),(5),(4 5)}, $4.next = 5.next = (4\ 5).next = 6$ et comme $4 \subset (4\ 5)$ et $5 \subset (4\ 5)$ les sommets 4 et 5 peuvent être supprimés. En revanche, $2.next = \{(3),(3\ 4),(3\ 4\ 5)\}$ mais $3.next \not\subset (3\ 4\ 5).next$ donc le sommet 3 n'est pas supprimé.
Le graphe utilisé lors de la vérification des candidats est illustré par la figure 5.9.

Le théorème 5 garantit que l'algorithme GTC_{ws}, adapté à la contrainte de windowSize, résoud le problème des chemins respectant minGap et windowSize.

Théorème 5 *L'algorithme GTC_{ws} construit exactement toutes les solutions de la plus grande taille possible pour le problème des chemins respectant minGap et windowSize.*

Démonstration : Nous avons déjà vu (Théorème 3) que l'inclusion de chemins de séquence (au sens de la définition 2) ne peut pas exister dans un graphe de séquence issu de la première version de GTC. Nous allons donc nous intéresser au seul cas de windowSize. Supposons que le graphe de séquence calculé par GTC_{ws} contienne deux chemins de séquence $s1$ et $s2$ tels que $s1 \subset s2$. cela voudrait dire

function GTC_{ws} **return** graphe de séquence
Input : une sous-séquence de données D
Output : $G(S,A)$ le graphe de séquence de D
foreach *itemset* $i \in D$ **do**
$S = S \cup \{i\}$
endforeach
$addWindowSize(S)$; //Ajoute à l'ensemble des sommets les combinaisons possibles avec windowSize
foreach $x \in S$ **do**
//Phase 1
$y = x$ **while** $y.begin() - x.begin() < minGap$ **do** $y{+}{+}$ $A = A \cup \{x,y\}$ $i_p = \{i \in S / i.begin() - y.begin() > minGap\}$ **foreach** $z \in i_p$ **do**
$z.isPrec[x] = 1$
endforeach
// Phase 2
$j_p = \{j \in S / j.begin() > y.begin()\ and\ j.isPrec[x] = 0\}$ **foreach** $t \in j_p$ **do**
$A = A \cup \{x,t\}$
endforeach
endforeach
$pruneIncluded(S,A)$; // Supprime du graphe les sommets faisant partie de séquences incluses.
end function GTC_{ws}

Algorithme 5.4: *GTC_{ws}, solution pour minGap & windowSize*

que le graphe de séquence contient le sous-graphe représenté par la figure 5.13. Or, la dernière phase de l'algorithme (cf. fonction $pruneIncluded(S,A)$) consiste pour chaque sommet x à déterminer l'ensemble de ses successeurs "$x.next$". pour chaque sommet y dans $x.next$, si $y \subset z$,$z \in x.next$ et $y.next \subseteq z.next$ le sommet y peut alors être retiré du graphe. Par construction, ce schéma ne peut pas apparaître dans le graphe issu de GTC$_{ws}$.

5.2 L'algorithme complet

L'algorithme 5.5 permet, pour une sous-séquence de données de déterminer quelles séquences candidates dans l'arbre des candidats sont incluses dans la sous-séquence de données concernée. L'algorithme général utilisé pour déterminer le nombre de passes qui seront effectuées reste le même (cf. algorithme 4.2 du Chapitre 4).

Pour prendre en compte la contrainte de maxGap dans l'algorithme GTC, nous devons considérer les estampilles des itemsets dans le graphe obtenu par GTC$_{ws}$. Un bref rappel sur maxGap nous dit que, d'après cette contrainte, un candidat n'est pas inclus dans une séquence de données s si il existe deux itemsets consécutifs dans c tels que l'écart entre l'estampille du premier itemset (appelé l_{i-1} dans la

procedure *verifyCandidate*
Input : L'arbre des candidats T. Une séquence de données d et son identifiant $idSeq$. La longueur des candidats à vérifier k. $minGap$, $maxGap$ et $windowSize$.
Output : L'arbre T vérifiant : $\forall c \in T / c \subseteq d$, $c.compteur$ est incrémenté.

```
g = GTC_ws(d,windowSize,minGap,maxGap)  foreach i ∈ racine.children do
    foreach S ∈ g do
        foreach j ∈ S.items do
            if i = j then
                recursiveVerifyCandidate(racine.children[i],S,i,g,0,minGap,maxGap,windowSize)
            endif
        endforeach
    endforeach
endforeach
end procedure verifyCandidate
```

Algorithme 5.5: *Vérification de candidats : niveau 1*

définition 5) et l'estampille du second itemset (appelé u_i dans la définition 5) dans s est supérieur à maxGap. D'après cette définition, quand on compare les candidats avec une séquence de données, on doit trouver dans un itemset du graphe, les estampilles de chaque item, dans la mesure où à cause de windowSize, les items peuvent être regroupés en un seul sommet du graphe. Afin de verifier maxGap, l'estampille de chaque sous-itemset correspondant aux itemsets inclus dans le sommet du graphe, doit vérifier maxGap avec l'itemset précédent et l'itemset suivant.

Considérons la base de données décrite par la figure 5.14. la figure 5.15 illustre alors le graphe obtenu en appliquant GTC_{ws} sur cette séquence avec une valeur de 1 pour windowSize et 0 pour minGap. Dans le but de déterminer si la séquence candidate < (2) (4 5) (6) > est incluse dans le graphe, nous devons examiner l'écart entre l'item 2 et l'item 5, ainsi que celui entre les items 4 et 6. Le principal problème auquel la comparaison se heurte consiste à retrouver les estampilles des items 3, 4 et 5 dans la mesure où les itemsets (3) et (4 5) ont été fusionnés en un seul itemset (3 4 5). Nous sommes donc conduits à déterminer les estampilles de chaque composants de cet itemsets hybride.

Client	Date	Items
C_1	01/01/2000	2
C_1	03/01/2000	3
C_1	04/01/2000	4 5
C_1	06/01/2000	6

FIG. 5.14 – *Base de données, illustrant le problème de maxGap*

Avant de présenter la façon dont GTC gère la contrainte maxGap, considérons que nous sommes en

procedure *recursiveVerifyCandidate*
Input : Le sommet N de l'arbre pour la validation en cours. L'arbre des candidats T. Un graphe de séquence g représentant une séquence de données d et son identifiant $idSeq$. La longueur des candidats à vérifier k, la longueur de candidats parcourus jusqu'ici (l). $minGap$, $maxGap$ et $windowSize$. $iPrec$ l'item précedemment vérifié dans le sommet du graphe $SPrec$.
Output : L'arbre T vérifiant : $\forall c \in T / c \subseteq d$, $c.compteur$ est incrémenté.
if $l = k$ *and* $N.lastSeq = idSeq$ **then**
  // Feuille non incrémentée par la séquence de données
  $N.lastSeq = idSeq \;\; N.cpt++$
else
  // Same transaction
  foreach $i \in SPrec/succ(i,iPrec)$ **do**
    // Les successeurs de i pour le sommet $SPrec$
    if $i \in N.same$ **then**
      $recursiveVerifyCandidate(N.same[i],SPrec,i,g,l+1,minGap,maxGap,windowSize)$
    endif
  endforeach
  // Other transaction
  foreach $S \in g/succ(S,Sprec)$ **do**
    foreach $i \in S.items$ **do**
      if $i \in N.other$ **then**
        $recursiveVerifyCandidate(N.other[i],S,i,g,l+1,minGap,maxGap,windowSize)$
      endif
    endforeach
  endforeach
endif
end procedure *recursiveVerifyCandidate*

Algorithme 5.6: *Vérification de candidats : niveaux 1 à k*

mesure de connaître les itemsets qui satisfont maxGap dans le graphe résultant. En utilisant cette information, la vérification des candidats peut alors être améliorée, comme l'illustre (toujours à l'aide de la séquence candidate < (2) (4 5) (6) >) l'exemple suivant.

Exemple 41 *Considérons le graphe de séquence décrit par la figure 5.15. Admettons que l'information concernant les sommets que l'on peut atteindre en fonction de maxGap soit disponible, et que maxGap soit fixé à une valeur de 4 jours. Considérons à présent la façon dont une inclusion est détectée entre un candidat et le graphe de séquence. L'itemset candidat (2) et l'itemset (2) du graphe de séquence sont tout d'abord comparés par l'algorithme. Comme maxGap est respecté et que* $(2) \subseteq (2)$*, alors le premier itemset de la séquence candidate est inclus dans le graphe de séquence et le processus peut continuer. Dans le but de vérifier les autres composants de la séquence candidate, nous devons savoir quel est le prochain itemset se terminant par 5 dans le graphe de séquence, et qui vérifie maxGap. En fait, en*

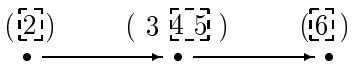

FIG. 5.15 – *graphe de séquence obtenu par* GTC_{ws}

considérant le dernier item du prochain itemset, si nous voulons savoir si la contrainte de maxGap est respectée entre l'itemset courant (2) et le prochain itemset de la séquence candidate, nous devons évaluer l'écart entre l'itemset courant dans le graphe et le prochain itemset terminé par 5 dans ce même graphe. Nous avons considéré que cette information est disponible dans le graphe. Cette information peut alors être prise en compte par l'algorithme afin d'atteindre directement l'itemset (3 4 5) *dans le graphe et le comparer avec l'itemset* (4 5) *de la séquence. Jusqu'à présent, le candidat est toujours inclus dans la séquence. Cependant, pour être complète, la comparaison doit chercher le prochain itemset du graphe se terminant par 6 et respectant maxGap (i.e. l'écart entre 4 et 6 doit être inférieur à 4 jours). Cette condition se vérifie avec le dernier itemset du graphe de séquence. À la fin du processus, nous pouvons conclure que c est incluse dans le graphe de séquence de d, ou plus précisemment que* $c \subseteq d$.

Considérons à présent le même exemple, mais avec une valeur de maxGap fixée à 2. Observons plus attentivement la deuxième itération. Comme nous considérons que l'information sur maxGap est disponible dans le graphe, nous savons qu'il n'y a aucun itemset se terminant par 5 et qui respecte maxGap avec l'item 2. Le processus se termine alors car la séquence ne peut pas être incluse dans la séquence de données, et le parcours du la structure du candidat peut cesser.

Décrivons à présent la façon dont l'information sur les itemsets qui vérifient maxGap est prise en compte par GTC. Chaque item, dans le graphe, est associé à un tableau qui indique les sommets que l'on peut atteindre, en fonction de maxGap. Chaque valeur du tableau est associée à un ensemble de sommets qui garantit que le sommet pointé correspond à un itemset qui se termine par cette valeur et que l'écart entre ces deux items est inférieur ou égal à maxGap. L'algorithme de vérification des candidats peut alors utiliser l'information du graphe pour déterminer les inclusions. La version finale de GTC est détaillé par l'algorithme 5.7.

Exemple 42 *Prenons l'exemple du graphe de séquence obtenu dans l'exemple 40 avec la base de données de la figure 5.8. Considérons que maxGap a pour valeur 2. Selon la discussion précédente, le graphe résultant est décrit par la figure 5.16. Examinons à présent l'itemset* (2). *D'après la valeur de maxGap, le sommet* (3 4 5) *peut être atteint depuis (2). Cependant, dans la mesure où la contrainte entre l'item 3 et l'item 6 n'est pas respectée, il n'y a pas d'itemset que l'on peut atteindre depuis l'item (3) et son tableau de valeurs associées est vide. D'un autre côté, d'après la valeur de maxGap, le sommet (3) peut être atteint depuis (2). Depuis ce sommet, les item 5 et 4 vérifient la contrainte de maxGap. Enfin, depuis les items 4 et 5 dans l'itemset* (4 5), *il est possible d'atteindre l'itemset* (6) *tout en respectant la contrainte de maxGap.*

function *GTC*
Input: d, une séquence de données
Output: G_d(S,A) le Graphe de séquence
G_d(S,A)=GTC$_{ws}$(d);
foreach *item* $i \in G_d(S,A)$ **do**
  foreach *item* $j \in G_d(S,A)$ **do**
    if *J.isPrec[i]=1 OU* $j \in$ *i.next* **then**
      addMax(i,j); // Ajouter j à la liste de pointeurs pour la valeur i du tableau de j
    endif
  endforeach
endforeach
end function *GTC*

Algorithme 5.7: *Version finale de GTC, prise en compte de maxGap*

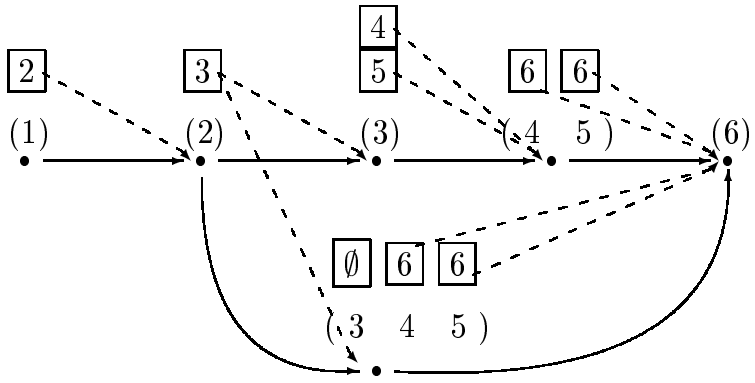

FIG. 5.16 – *graphe de séquence obtenu par GTC (version finale)*

5.3 Optimisations

Comme nous venons de le voir, l'intégration de la contrainte de maxGap au sein même du graphe de séquence peut améliorer le processus de vérification des candidats. Afin de profiter de cette information, l'algorithme de vérification des candidats doit atteindre le dernier item de chaque itemset de la séquence candidate de manière efficace. Cependant les structures mises en place à l'heure actuelle, comme celles de PSP ou GSP, ne permette pas une telle efficacité sur l'accès au dernier item (un parcours de toute la branche de l'itemset est nécessaire). Considérons par exemple la façon dont le dernier item d'un itemset est atteint par l'algorithme PSP. Comme les séquences candidates sont stockées dans un arbre préfixé, la navigation dans l'arbre se produit du sommet courant vers un sommet N tel que $\exists M/M \in fils(N)$ et $M \notin itemset(N)$. Cette navigation est très coûteuse et se voit répétée autant de fois qu'il y a d'itemset dans la séquence de données.

Cette section a pour but de présenter une amélioration que l'on peut apporter aux structures pour être

avantagée par l'information disponible sur maxGap dans les graphes de séquence. Considérons tout d'abord la structure d'arbre préfixé proposé par l'approche PSP. Ensuite, les optimisations pour GSP seront proposées.

Optimisations pour l'algorithme PSP

Considérons l'arbre préfixé utilisé par l'algorithme PSP, avec les séquences candidates suivantes :

< (1) (2 3 4) (5) >
< (1) (3 4) (5) (6) >
< (1) (2 3) (4) (5)>

qui sont organisées selon la structure d'arbre préfixé décrite par la figure 5.17.

Afin de profiter de l'information fournie par le graphe de séquence, nous proposons d'améliorer cette structure de la façon suivante. Dans la mesure où le principal problème consiste à trouver le dernier item dans le prochain itemset de façon efficace, nous proposons de considérer une séquence candidate comme suit : chaque itemset dans la séquence candidate est inversé, mais les itemsets gardent leur ordre d'apparition, en respectant leur estampillage. À partir de cette solution, l'ensemble des candidats devient :

< (1) (4 3 2) (5) >
< (1) (4 3) (5) (6) >
< (1) (3 2) (4) (5)>

et la structure arborescente qui leur est associée est décrite à la figure 5.18. Cette nouvelle écriture de l'arbre mis en place par PSP demande un temps d'exécution minime, mais entraîne des changements dans l'algorithme de parcours des candidats. La comparaison des items sera effectuée comme décrit par les figures 5.19 et 5.20.

Pour illustrer cette comparaison, considérons le graphe de séquence de la figure 5.19. Tout d'abord, le premier item du graphe (i.e. (1)) est comparé avec la racine du l'arbre des candidats proposés par PSP (i.e. (1)). Comme l'inclusion entre les deux itemsets est vérifiée, le processus peut continuer. Le prochain item de la structure candidate est 4. Comme nous pouvons constater que 4 est le dernier item vérifiant maxGap dans le prochain itemset (2 3 4) du graphe de séquence, une comparaison est faite entre l'item précédent 4 (i.e. 3) et les prochain item de la structure d'arbre préfixé (i.e. 3). En suivant cette navigation l'item 2 dans l'arbre et l'item 2 dans le graphe sont comparés avec succès. En raison d'un changement d'itemset, le prochain itemset de l'arbre est (5). À partir de ce point, nous devons trouver dans le graphe, le prochain itemset se terminant par 5 et respectant un écart inférieur à maxGap par rapport à l'item courant (i.e. 2). Comme cette information est disponible dans le graphe, la comparaison est faite entre le prochain item dans l'arbre (i.e. 5) et le prochain item dans le graphe de séquence (i.e. 5). À la fin de ce parcours, nous pouvons conclure que la séquence candidate < (1) (2 3 4) (5) > est incluse dans le graphe de séquence.
D'après le parcours illustré par la figure 5.20, nous pouvons également conclure que la séquence candidate < (1) (3 4) (5) (6) > est incluse dans ce graphe de séquence.

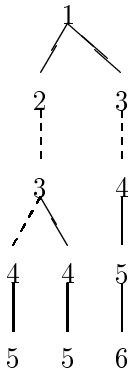

FIG. 5.17 – *Candidats gérés par la structure préfixée classique*

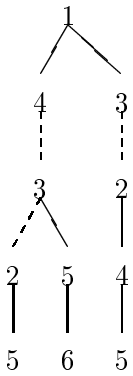

FIG. 5.18 – *Candidats gérés par la structure préfixée avec inversion des itemsets*

Optimisations pour l'algorithme GSP

Lors du parcours de l'arbre de candidats proposés par GSP (structure d'arbre de hachage), le changement d'itemset ne peut être pris en compte. Cette information n'est disponible qu'une fois les feuilles de l'arbre atteintes, rendant alors inutile l'information sur maxGap durant le parcours de l'arbre. Cependant, une fois les feuilles susceptibles de contenir des candidats inclus dans la séquence de données atteintes, il est possible de réorganiser les candidats qu'elles contiennent. En effet il suffit d'inverser le contenu des itemsets de ces candidats (tout en gardant l'organisation des itemsets entre eux) afin de tirer parti de l'information disponible sur maxGap et envisager des parcours de séquences tels que ceux décrits par les figures 5.19 et 5.20, présentés dans l'optimisation proposée pour PSP.

5.4 Expérimentation

Pour comparer les performances des deux algorithmes mis en jeu (GTC et PSP) et dans le but de compléter l'étude sur le comportment des deux algorithmes, nous avons généré de nouveaux jeux de

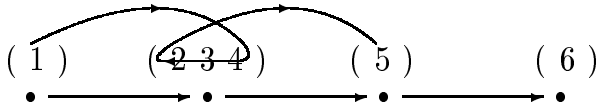

FIG. 5.19 – *Navigation dans le graphe avec inversion des itemsets*

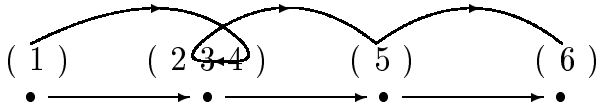

FIG. 5.20 – *Navigation dans le graphe avec inversion des itemsets*

données synthétiques. A l'aide du même générateur de données, utilisé pour les tests décrits à la section 4.4, nous avons généré des jeux de donnés moins volumineux. Dans la mesure où le but est de comparer les deux algorithmes, leurs comportements gardent des écarts dans des proportions égales en fonction de la taille des benchmarks utilisés. La figure 5.21 résume les paramètres utilisés pour générer les deux fichiers de test. Le fichier Access-Log est le même depuis la section 4.4.
Cependant, pour analyser le comportement de GTC sur des bases de données de grande taille et ainsi obtenir une meilleure connaissance de ses capacités, nous avons effectué des tests dont les résultats sont donnés dans la section 5.5.
Nous pouvons constater, après les series de tests effectués, que la reflexion proposée en début de chapitre se vérifie : PSP teste trop de possibilités à partir d'une séquence de données et avec des contraintes de minGap et de windowSize non nulles. En effet, plus windowSize augmente, plus PSP effectue d'appels récursifs. Malgré un comportement linéaire par rapport à l'augmentation de windowSize, PSP affiche des temps de réponse moins performants que GTC sur les benchmarks testés et avec l'utilisation des contraintes de temps. Un aspect intéressant des test pratiqués réside dans la quantité d'appels récursifs relevée pour chaque algorithme. Nous pouvons en effet constater que, pour chaque benchmark, une corrélation est observable entre les temps de réponse et le nombre d'appels récursifs effectués (une courbe d'un benchmark représentant les temps de réponse a exactement la même allure que celle dénombrant les appels récursifs pour ce même benchmark).
Il est également important de constater que, pour ces benchmarks, les fréquents sont volontairement de grande taille. Nous avons donc généré des instances de grand potentiel, et les complexités des algorithmes étant dépendantes de la taille de la réponse, nous avons une confirmation de la bonne conduite de ce projet : GTC est une solution particulièrement efficace pour des instances de grand potentiel

Nom	$\|D\|$	$\|C\|$	$\|S\|$	$\|N\|$
C20-D1-S10-N1	1000	20	10	1000
C20-D1-S8-N1	1000	20	4	1000

FIG. 5.21 – *Paramètres des fichiers de test*

intégrant des contraintes de temps.

5.5 Discussion

GTC est une solution efficace pour résoudre le problème de la recherche de séquences généralisées. En effet, pour des problèmes comme la recherche de motifs séquentiels, PSP est globalement plus rapide que GTC d'une faible constante. Cette différence s'explique par le coût de la construction du graphe, qui est mis en œuvre de manière inutile quand aucune contrainte de temps n'est demandée (cf. figure 5.26).

L'algorithme de constrution du graphe semble être un investissement rentable quand la longueur des candidats à vérifier est grande. Il est, en effet, immédiat de constater que le temps de construction du graphe doit être inférieur au temps de parcours de PSP diminué du temps de parcours de GTC. Or, le temps nécessaire pour parcourir l'arbre des séquences candidates avec une séquence de données dans PSP croît de façon bien plus rapide que le temps nécessaire pour parcourir l'arbre des séquences candidates avec un graphe de séquence, alors que le temps de construction du graphe de séquence, lui, reste constant. Ces différences de temps requis sont, de plus, proportionnelles à l'importance des contraintes de temps spécifiées par l'utilisateur. Nous avons pourtant constaté que pour des contraintes de temps assez faible (windowSize=1 ou bien minGap=0) ou pour des fréquents relativement courts, la seconde passe sur la base de données (destinée à tester les candidats de taille 2) restait encore plus efficace avec l'algorithme PSP. Malgré cela, GTC affiche des temps de réponse plus performants que PSP dans de tels cas de figure (grace à une plus grande rapidité d'exécution sur les passes suivantes). Pour exploiter ces comportements, nous avons crée un troisième algorithme (PG-*Hybrid*), hybride, utilisant la technique de PSP pour la seconde passe et ensuite GTC sur les passes suivantes. Comme le montrent les résultats illustrés par la figure 5.27, cette technique s'avère efficace pour des instances de faible potentiel (des fréquents courts), alors que pour des fréquents longs la différence de temps devient minime en proportion du temps total requis.

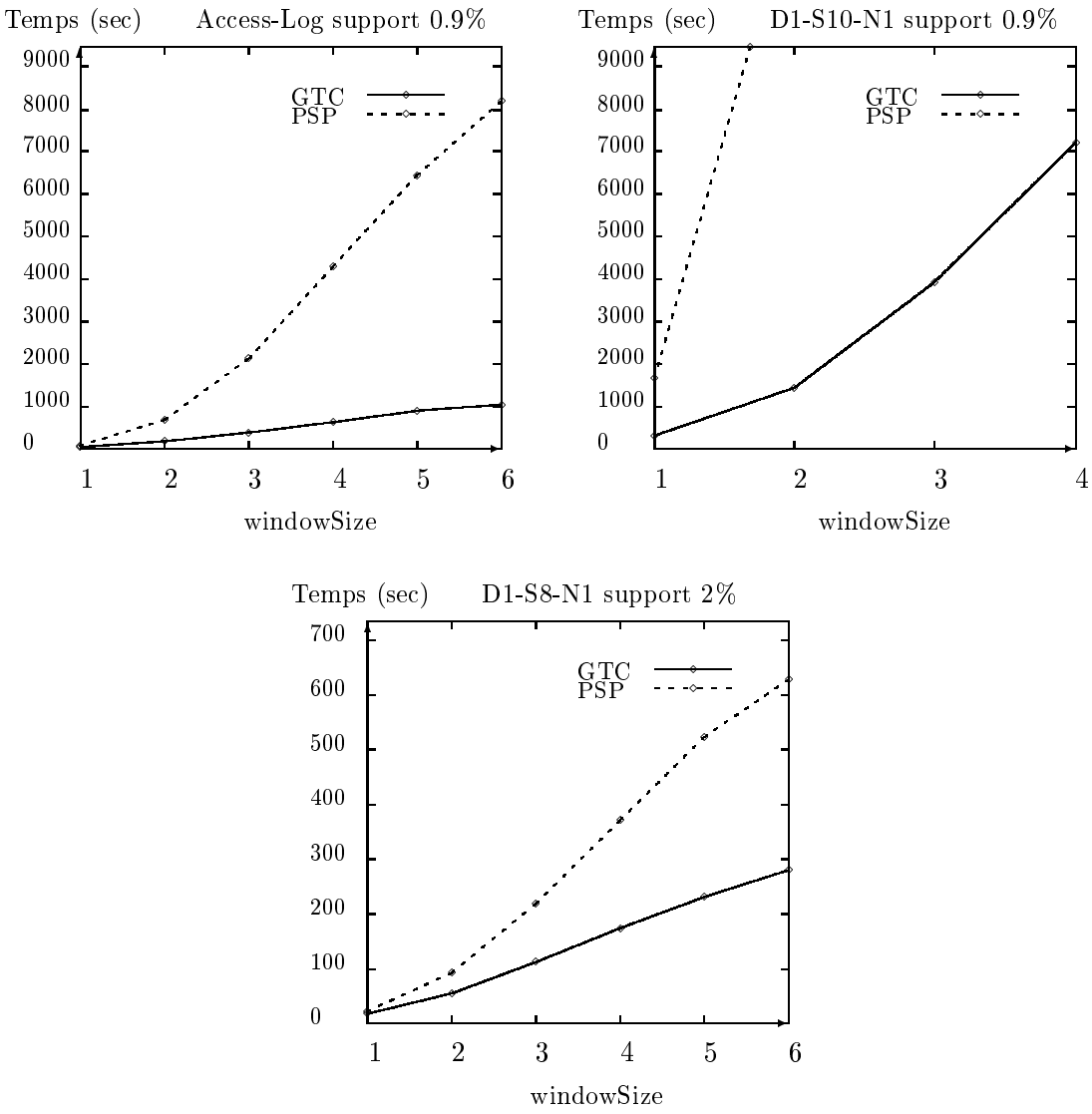

FIG. 5.22 – *Temps d'exécution comparés de PSP et GTC sans minGap*

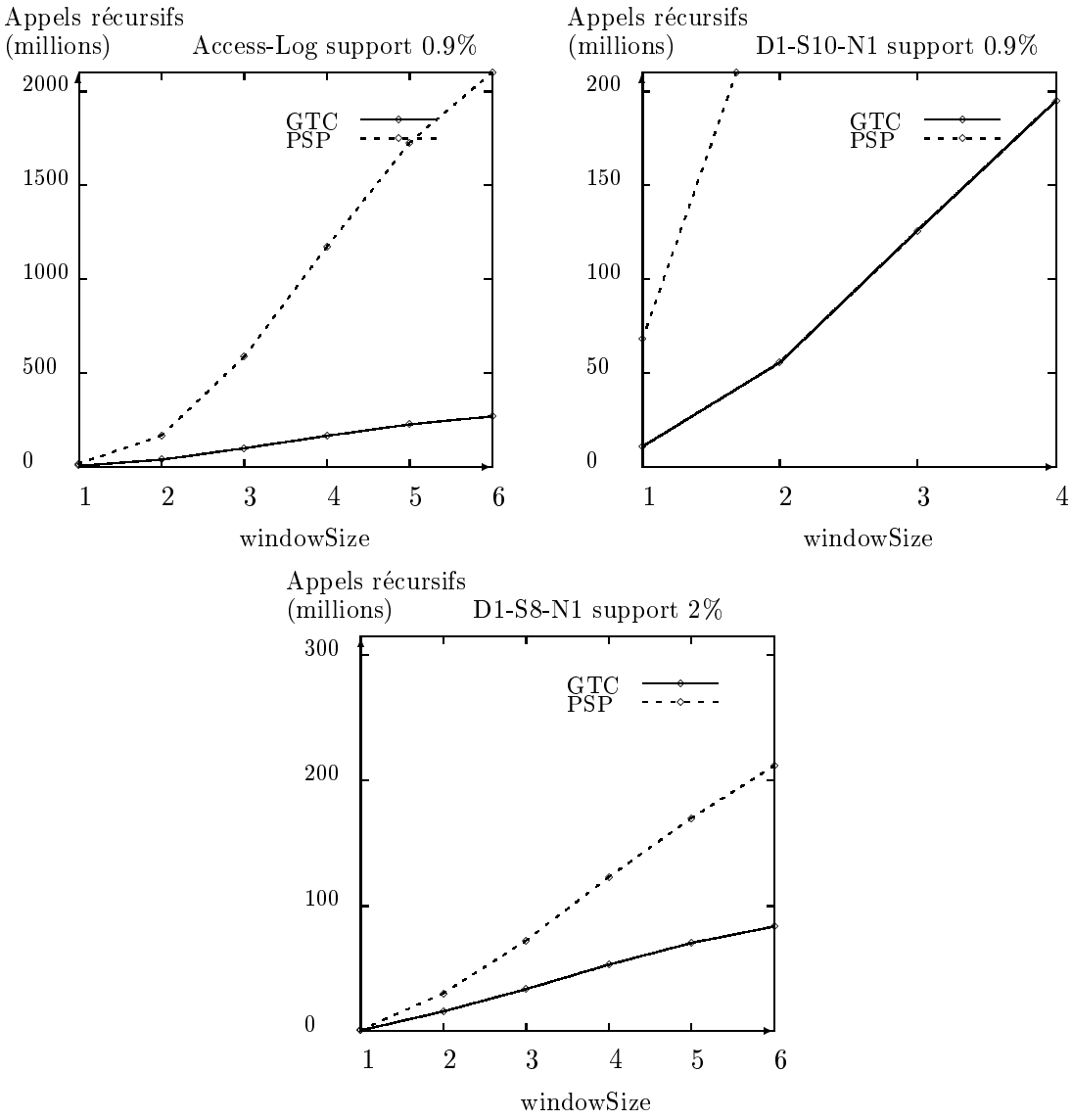

FIG. 5.23 – *Nombre d'appels récursifs comparés de PSP et GTC sans minGap*

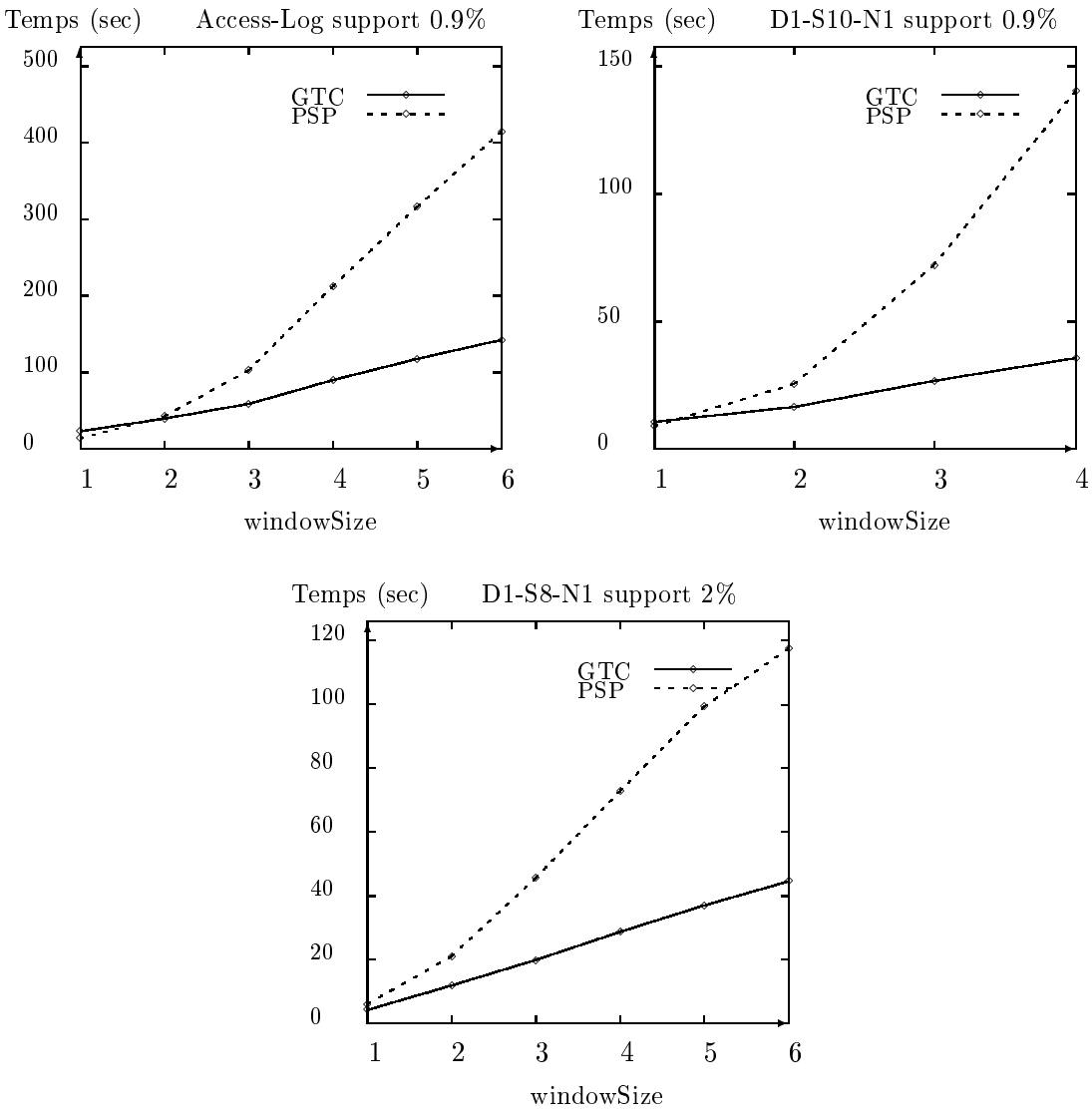

FIG. 5.24 – *Temps d'exécution comparés de PSP et GTC avec minGap*

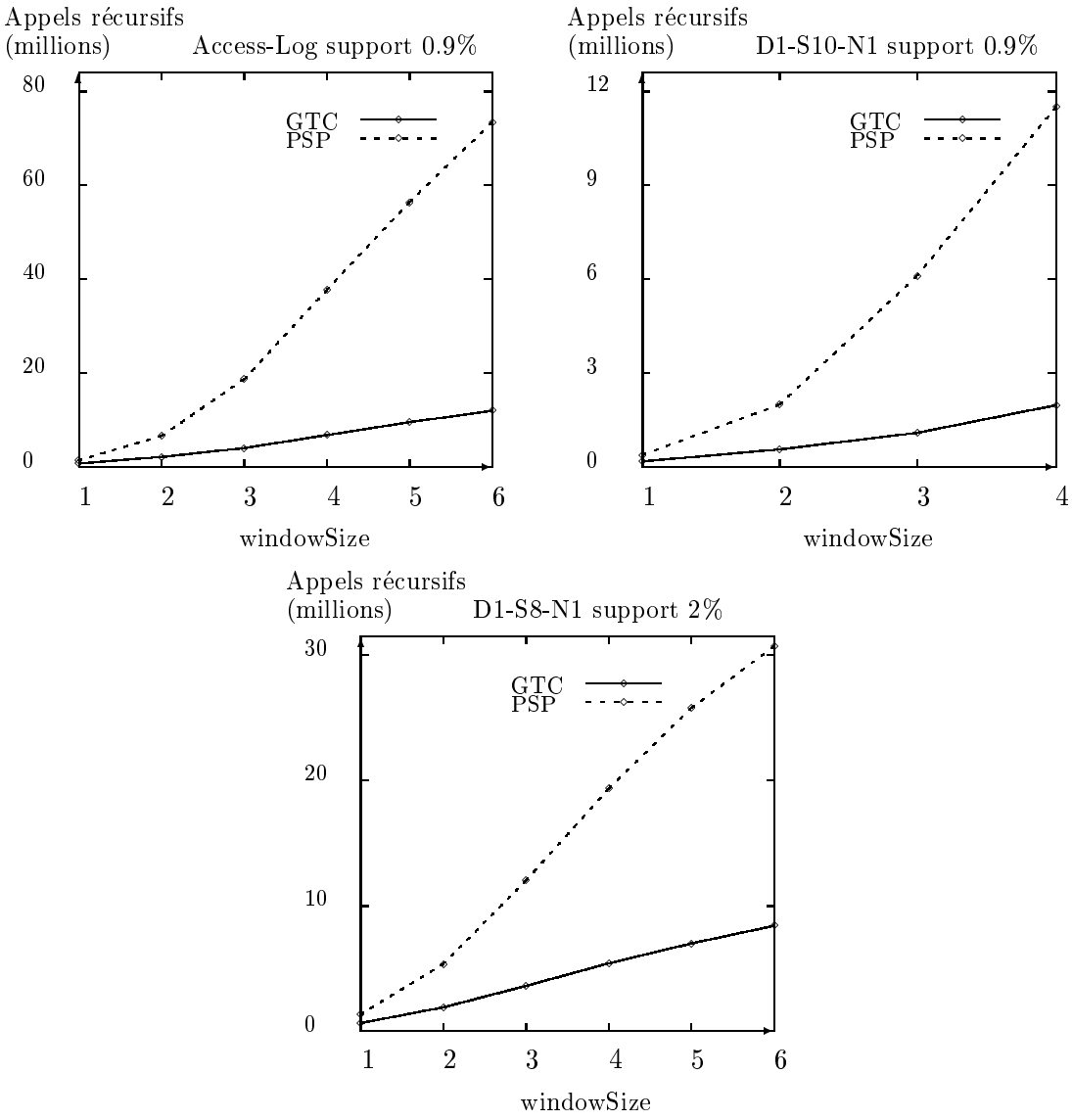

FIG. 5.25 – *Nombre d'appels récursifs comparés de PSP et GTC avec minGap*

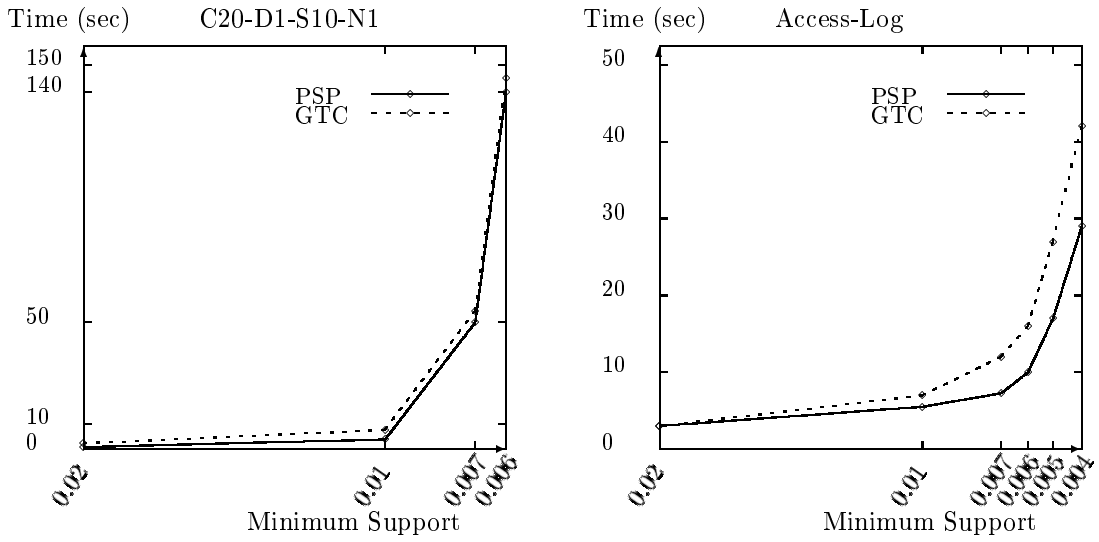

FIG. 5.26 – *Temps d'exécution comparés de PSP et GTC, windowSize=0*

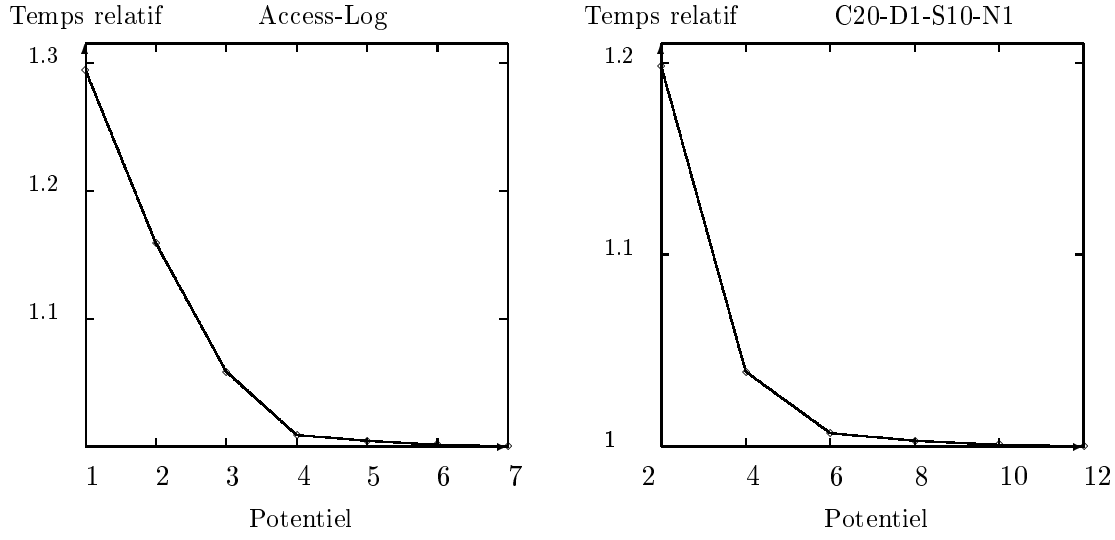

FIG. 5.27 – *Temps d'exécution relatif de GTC par rapport à hybrid, windowSize=1*

Chapitre 6

Extraction incrémentale de motifs séquentiels

Client	Itemsets		
$C1$	10 20	20	50 70
$C2$	10 20	30	40
$C3$	10 20	40	30
$C4$	60	90	

(DB)

Itemsets	
50 60 70	*80 100*
50 60	*80 90*

(db)

FIG. 6.1 – *Une base de données (DB) et l'incrément (db) contenant de nouvelles transactions*

Client	Itemsets		
$C1$	10 20	20	50 70
$C2$	10 20	30	40
$C3$	10 20	40	30
$C4$	60	90	
$C5$			

(DB)

Itemsets	
50 60 70	*80 100*
50 60	*80 90*
10 40	*70 80*

(db)

FIG. 6.2 – *Une base de données (DB) et son incrément constitué de nouvelles transactions et de nouveaux clients (db).*

Ce chapitre est destiné à présenter les travaux que nous avons menés dans le cadre de l'extraction incrémentale de motifs séquentiels. Nous y expliquons le fonctionnement de l'algorithme ISE que nous avons développé dans cette optique. Nous présentons tout d'abord la méthode du point de vue des ensembles manipulés (section 6.1.1), puis nous détaillons l'algorithme lui-même (section 6.1.2), avant de présenter les optimisations sur le nombre de candidats générés (section 6.3). Enfin une discussion sur l'utilisation d'ISE hors du contexte incrémental est proposée dans la section 6.5.

6.1 ISE : un algorithme incrémental pour l'extraction de motifs séquentiels

Considérons à nouveau l'exemple donné en introduction pour illustrer la problématique liée à l'extraction incrémentale de motifs séquentiels. Cet exemple servira d'illustration tout au long de ce chapitre, afin de préciser certaines spécificités de notre méthode. Nous avons distingué dans cet exemple deux cas qui, par ordre de complexité, pouvaient se présenter dans le cadre de l'ajout de données dans une base. Le premier cas (illustré par la figure 6.1) concerne l'ajout de transactions à des clients existants dans la base de données, alors que le second cas (illustré par la figure 6.2) est une extension de cette problématique, qui prend en compte l'ajout de nouveaux clients et de nouvelles transactions.

6.1.1 Vue d'ensemble

Pour résoudre la problématique de l'extraction incrémentale de motifs séquentiels, ISE manipule trois ensembles de séquences. Tout d'abord, les séquences de DB qui peuvent devenir fréquentes si elles apparaissent dans db. Ensuite, les séquences qui n'apparaissent pas sur DB mais qui sont fréquentes sur db. Enfin, les séquences fréquentes sur U qui n'appartiennent à aucun des deux ensembles précédents (i.e. supportées par au moins une séquence de données, partagée entre DB et db).Le tableau 6.1 récapitule les notations utilisées dans l'algorithme ISE.

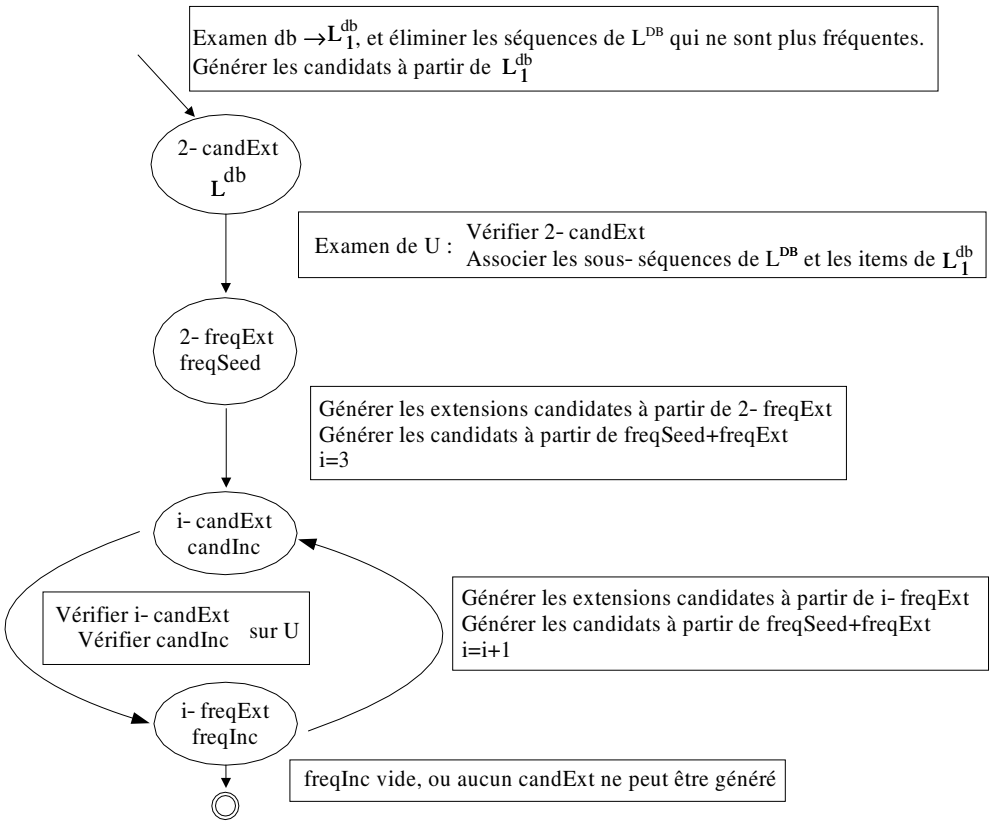

FIG. 6.3 – *La gestion des trois ensembles de séquences par* ISE

La figure 6.3 illustre la façon dont ces ensembles sont gérés. La première passe de ISE se situe sur db et permet d'identifier les items appartenant à db, ainsi que le nombre de leurs apparitions sur db et donc sur U (par addition avec cette information sur DB). On peut alors, à l'issue de cette passe, déterminer les items fréquents sur U qui apparaissent dans db : L_1^{db}. A partir de cet ensemble nous construisons les séquences de taille 2 ($L_1^{db} \times L_1^{db}$), qui apparaissent sur db (nouvelle passe sur db). Ces séquences forment l'ensemble 2-$candExt$.
Ces extensions sont alors vérifiées par une première passe sur U pour donner l'ensemble 2-$freqExt$. ISE

L^{DB}	Séquences fréquentes sur DB.
L_1^{db}	Séquences fréquentes de taille 1 sur db et validées sur U.
$candExt$	Séquences candidates générées à partir de db.
$freqExt$	Séquences de $candExt$ validées sur U.
$freqSeed$	Séquences fréquentes sur U, de la forme $<$(sous-séquence de L^{DB}) (séquence de L_1^{db})$>$.
$candInc$	Séquences candidates générées à partir de $freqSeed$ et $freqExt$.
$freqInc$	Séquences de $candInc$ validées sur$U$.
L^U	Séquences fréquentes sur U.

TAB. 6.1 – *Notations pour l'algorithme* ISE

profite de cette passe pour associer aux sous-séquences de L^{DB} les items fréquents de L_1^{db}, et vérifier la validité de ces associations (i.e. la sous-séquence s de L^{DB} est elle fréquente si on lui ajoute l'item i de L_1^{db} ?). Les associations validées sur U permettent d'obtenir l'ensemble $freqSeed$ qui constitue la base des candidats qui seront désormais générés.

Les passes supplémentaires sont basées sur la méthode suivante : nous disposons de deux ensembles, 2-$freqExt$ et $freqSeed$. À partir de 2-$freqExt$ l'ensemble 3-$candExt$ est généré. A partir des ensembles 2-$freqExt$ et $freqSeed$, l'ensemble $candInc$ est généré selon le principe suivant : soit $s1 \in freqSeed$ et $s2 \in freqExt$, si le dernier item de $s1$ est égal au premier de $s2$, alors générer la séquence $s3$ en supprimant le premier item de $s2$ et en concaténant $s1$ et $s2$, enfin ajouter $s3$ à $candInc$. Il faut alors vérifier ces deux ensembles ($candExt$ et $candInc$) sur U.

Ce procédé est répété jusqu'à ce que l'on ne puisse plus générer d'extension candidate dans $candExt$ ou de séquences dans $candInc$. À la fin de ce processus, nous obtenons les séquences fréquentes sur U, à partir des séquences de L^{DB} et des séquences maximales de $freqSeed \bigcup freqInc \bigcup freqExt$.

La figure 6.4 donne un autre point de vue sur ce processus. La première passe sur db nous permet d'obtenir les extensions fréquentes. La combinaisons des items fréquents sur db et des fréquents sur DB aboutit sur l'ensemble $candSeed$. Les extensions fréquentes combinées avec elles-mêmes permettent d'obtenir l'ensemble 2-$candExt$ (les extensions candidates). Partant de ces deux ensembles ($candSeed$ et 2-$candExt$, les deux cellules du haut à gauche et à droite dans la figure 6.4) le processus peut démarrer. Une passe sur U permet de détecter les extensions fréquentes et les candidats à l'incrémental devenus fréquents. L'ensemble 2-$candExt$ est filtré pour donner l'ensemble 2-$freqExt$. L'ensemble $candSeed$ est filtré pour donner l'ensemble $freqSeed$. Ensuite le principe admet un aspect répétitif. La combinaison de 2-$freqExt$ et de $freqSeed$ donne l'ensemble $candIncr$ alors que la combinaison de 2-$freqExt$ avec lui-même donne l'ensemble 3-$candExt$. Le principe de la passe sur U est alors réitéré et le processus continue jusqu'à ce qu'aucun candidat ne soit généré ou aucun nouveau fréquent ne soit découvert.

Dans la mesure où l'ajout de nouveaux clients a pour principale conséquence une vérification du support des séquences fréquentes sur DB, nous illustrons par la suite la partie de l'algorithme qui concerne l'ajout de transactions à des clients existants. Enfin, l'exemple 47 illustre le comportement d'ISE quand

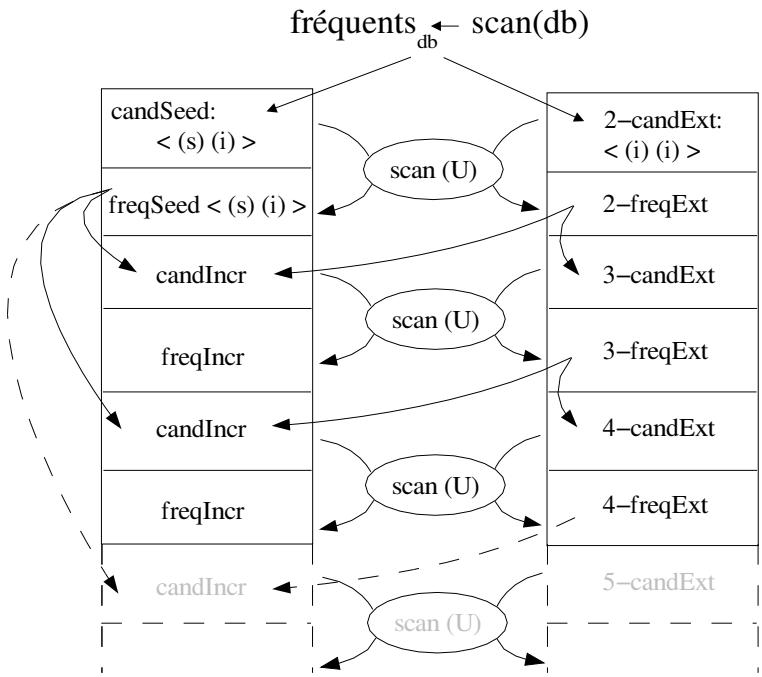

FIG. 6.4 – *La méthode proposée, vue d'ensemble*

de nouvelles transactions et de nouveaux clients sont ajoutés à la base DB.

6.1.2 Les étapes en détails

Première passe

Au cours de la première passe sur db, nous associations à chaque item son support sur db. Pour cela nous procédons de la manière suivante : pour chaque client $C \in db$, nous estimons grâce à un parcours de la transaction de C le nouveau support des items de C apparaissant sur la partie de la transaction de C qui se limite à db. Ainsi, si l'item $i \in db$ pour le client C n'apparait pas déjà dans la transaction de C alors nous incrémentons son compteur. A la fin de cette passe et en confrontant ce résultat aux fréquences des items sur DB nous sommes en mesure de déterminer quels items de db sont fréquents sur U. Cet ensemble de base est appelé L_1^{db}.

En outre, au cours de cette passe sur db nous examinons les nouveaux clients, pour déterminer si les séquences de L^{DB} sont vérifiées et avec quel pourcentage. De cette façon, à la fin de cette passe, nous

supprimons de L^{DB} les séquences qui ne vérifient plus le support minimum.

Exemple 43 *Considérons la base de données incrément de la figure 6.1. Après une passe sur db, les supports des items apportés par la mise à jour sont:* $\{(< (50) >, 2), (< (60) >, 2), (< (70) >, 1), (< (80) >, 2), (< (90) >, 1), (< (100) >, 1)\}$. *Considérons maintenant qu'après la phase de fouille de données précédente sur DB, les supports des items sur DB soient les suivants:*

item	10	20	30	40	50	60	70	90
support	3	3	2	2	1	1	1	1

En combinant ces items avec le résultat de la passe sur db, nous obtenons l'ensemble des items fréquents (les séquences fréquentes de taille 1) de U apparaissant sur db : $L_1^{db} = \{< (50) >, < (60) >, < (70) >, < (80) >, < (90) >\}$.

A partir de cet ensemble nous construisons les séquences de taille 2 ($L_1^{db} \times L_1^{db}$), qui apparaissent sur *db* (nouvelle passe sur *db*). Ces séquences forment l'ensemble 2-$candExt$. Cette phase diffère des passes sur la base de données effectuées par des méthodes comme GSP, dans la mesure où nous ne tenons pas compte du support (une seule apparition de la séquence s suffit pour que s soit placée dans 2-$candExt$). En effet, d'après le Lemme 2 (Cf. Section 6.2), une séquence candidate de taille 2 appartient a *2-candExt* si et seulement si, elle apparaît au moins une fois dans *db*. Nous ne voulons donc pas tester sur U l'ensemble des candidats réalisables à partir de L_1^{db} (i.e.$L_1^{db} \times L_1^{db}$) mais seulement l'ensemble des extensions qui peuvent être fréquentes sur U. En d'autres termes, si une séquences candidate de taille 2 n'apparaît pas sur *db* alors il est impossible de la retrouver comme l'extension d'une séquence fréquente de DB qui fera de la séquence finale une séquence fréquente sur U.

L'étape suivante consiste à analyser U, pour déterminer quelles séquences de *2-candExt* sont effectivement fréquentes. L'ensemble de séquences fréquentes de taille 2 ainsi obtenu est appelé $freqExt$.

Exemple 44 *Considérons l'ensemble d'items fréquents de l'exemple précédent :* L_1^{db}. *À partir de cet ensemble, nous pouvons générer les séquences suivantes* $< (50\ 60) >$, $< (50)\ (60) >$, $< (50\ 70) >$, $< (50)\ (70) >$, ..., $< (80)\ (90) >$. *Parmi ces séquences, considérons la séquence* $< (50)\ (60) >$. *Cette séquence n'apparaît pas sur db, elle ne peut donc pas être fréquente sur U et ne sera pas prise en compte. À la fin de la passe sur U, les séquences fréquentes de taille 2 sont:* $2\text{-}freqExt = \{< (50\ 60) >, < (50)\ (80) >, < (50\ 70) >, < (60)\ (80) >\}$.

Au cours de cette passe sur U, une seconde opération est effectuée, en fonction des items découverts fréquents sur *db*. Ce calcul, basé sur la propriété suivante et sur le Lemme 2 associe à chaque sous-séquence fréquente s de L^{DB} et à chaque item i de L_1^{db}, le nombre de fois où s précède i.

propriété 4 *Soient deux séquences A et B telles que* $A \subseteq B$, *alors* $supp(A) \geq supp(B)$ *car toute séquence qui supporte B dans DB supporte également A.*

Dans le but de trouver rapidement les sous-séquences de L^{DB} qui précèdent un item de L_1^{db}, nous utilisons pour chaque sous-séquence, un tableau qui possède autant d'éléments que le nombre d'items dans L_1^{db}. Au cours de la passe sur U, pour chaque séquence de données d et pour chaque sous-séquence s nous vérifions si $s \subseteq d$. Si tel est le cas alors le support de chaque item suivant la sous-séquence est

incrémenté.

Au cours de cette passe, qui vise à déterminer l'ensemble *2-freqExt*, nous obtenons aussi l'ensemble des sous-séquences fréquentes précédant des items de L_1^{db}. Cet ensemble nous permet alors d'obtenir un nouvel ensemble de séquences fréquentes de taille $j \leq (k+1)$, avec k la taille des fréquents obtenus lors de la précédente phase de fouille de données. Cet ensemble, appelé $freqSeed$ est construit de la manière illustrée par l'exemple suivant :

Items	Sous-séquences fréquentes
50	$< (10) >_3$ $< (20) >_3$ $< (30) >_2$ $< (40) >_2$ $< (10)\ (30) >_2$ $< (10)\ (40) >_2$ $< (20)\ (30) >_2$ $< (20)\ (40) >_2$ $< (10\ 20) >_3$ $< (10\ 20)\ (30) >_2$ $< (10\ 20)\ (40) >_2$
60	$< (10) >_2$ $< (20) >_2$ $< (30) >_2$ $< (40) >_2$ $< (10)\ (30) >_2$ $< (10)\ (40) >_2$ $< (20)\ (30) >_2$ $< (20)\ (40) >_2$ $< (10\ 20) >_2$ $< (10\ 20)\ (30) >_2$ $< (10\ 20)\ (40) >_2$
70	$< (10) >_2$ $< (20) >_2$ $< (10\ 20) >_2$
80	$< (10) >_2$ $< (20) >_2$ $< (30) >_2$ $< (40) >_2$ $< (10)\ (30) >_2$ $< (10)\ (40) >_2$ $< (20)\ (30) >_2$ $< (20)\ (40) >_2$ $< (10\ 20) >_2$ $< (10\ 20)\ (30) >_2$ $< (10\ 20)\ (40) >_2$
90	-

FIG. 6.5 – *Sous-séquences fréquentes associées aux items de* L_1^{db}.

Exemple 45 *Considérons l'item* 50 *de* L_1^{db}. *Pour le client* C_1, 50 *est précédé par les sous-séquences fréquentes suivantes :* $< (10) >$, $< (20) >$ *et* $< (10\ 20) >$. *Si nous considérons le client* C_2, *en tenant compte de la mise à jour, alors l'ensemble de sous-séquences fréquentes qui précèdent* 50 *devient :* $< (10) >$, $< (20) >$, $< (30) >$, $< (40) >$, $< (10\ 20) >$, $< (10)\ (30) >$, $< (10)\ (40) >$, $< (20)\ (30) >$, $< (20)\ (40) >$, $< (10\ 20)\ (30) >$ *et* $< (10\ 20)\ (40) >$. *Ce processus se réitère jusqu'à ce que toutes les transactions soient examinées. La figure 6.5, recense toutes les associations faites entre items et sous-séquences fréquentes sur* U.
Observons maintenant l'item 90. *Même si la séquence* $< (60)\ (90) >$ *peut être détectée pour les clients* C_3 *et* C_4, *elle ne peut être prise en considération dans la mesure où* 60 *n'était pas un item fréquent dans la base de données d'origine, i.e.* $60 \notin L^{DB}$. *En fait cette séquence sera découverte dans la prochaine phase d'exploration sur* U.
L'ensemble $freqSeed$ *est obtenu par concaténation de chaque item de* L_1^{db} *avec les sous-séquences respectant le support minimum qui leur sont associées. Par exemple, l'item* 70 *sera associé aux sous-séquences suivantes :* $< (10) >$, $< (20) >$ *et* $< (10\ 20) >$. *Nous retrouverons donc (en ce qui concerne* 70*) dans* $freqSeed$*:* $< (10)\ (\mathbf{70}) >$, $< (20)\ (\mathbf{70}) >$ *and* $< (10\ 20)\ (\mathbf{70}) >$.

Une fois cette première passe sur U terminée, nous obtenons donc un nouvel ensemble de fréquents de taille 2 (i.e. *2-freqExt*) ainsi qu'un nouvel ensemble de séquences fréquentes de taille inférieure ou égale

à k+1 (i.e. *freqSeed*).
Les passes suivantes nous permettent, à partir des ensembles énumérés dans cette section, de reproduire un processus de fouille de données complet, aboutissant à la découverte des fréquents sur U.

j^{eme} passe (avec $j > 1$)

Plaçons nous à présent dans le cadre d'une passe j, avec $j > 1$. La première des opérations consiste à générer des candidats à partir des deux ensembles de base trouvés dans la première passe sur U et décrite dans la section 6.1.2. Le principe de cette génération consiste à trouver parmi les séquences de $freqSeed$ et j-*freqExt*, tous les couples de séquences ($s \in freqSeed$, $s' \in j\text{-}freqExt$) tels que le dernier item i de s vérifie les deux conditions suivantes :

- $i \in L_1^{db}$
- i est le premier item de s'.

La première condition étant implicite, compte tenu des propriétés des items de s', elle est en fait une optimisation de la phase de recherche de ces couples de séquences. Lorsqu'un tel couple est découvert, la génération se produit en supprimant le dernier item de s et en concaténant s' à la séquence résultant de cette suppression. L'ensemble de séquences candidates ainsi généré est appelé $(j+1)$-$candInc$.

Une seconde opération est effectuée à partir de j-$freqExt$, en plus de la génération décrite ci-avant. Cette opération consiste à générer, selon le principe de génération de GSP, de nouveaux candidats de taille $i+1$ à partir des séquences de i-$freqExt$ (avec i la taille des extensions découvertes sur *db*). Les candidats ainsi générés, et qui apparaissent au moins une fois sur *db*, sont alors insérés dans l'ensemble $(i+1)$-$candExt$. les supports des séquences de cet ensemble sont alors évalués au cours de la passe sur U afin de déterminer lesquelles sont fréquentes. Les deux ensembles ainsi analysés (i.e. $(j+1)$-$candInc$ et $(i+1)$-$candExt$) deviennent alors (après suppression des non fréquents) $freqInc$ et $(j+1)$-$freqExt$. Ces deux nouveaux ensembles sont alors utilisés lors des générations de candidats suivantes et ce processus se réitère jusqu'à ce qu'aucun candidat ne puisse plus être généré.
À la fin de cette phase, L^U est obtenu en gardant les séquences maximales de
$L^{DB} \bigcup freqSeed \bigcup freqInc \bigcup freqExt$.

Exemple 46 *Reprenons notre exemple. Nous savons qu'à la fin de la première étape $k = 3$ (i.e. la plus longue séquence fréquente découverte est de taille 3). A la fin de la première passe, l'ensemble 2-$freqExt$ contenait (entre autres) les séquences* $<$ (50 60) *et* $<$ (60) (80) $>$. *Nous pouvons alors générer grâce à $freqExt$ un nouveau candidat :* $<$ (50 60) (80) $>$.
À présent, observons la génération de séquences candidates grâce à $freqSeed$ and 2-$freqExt$. Considérons les séquences $s =<$ (20) (40) (50) $>$ de $freqSeed$ et $s' =<$ (50 60) $>$ de 2-$freqExt$. Un nouveau candidat est obtenu en supprimant 50 *de s et en ajoutant s' au résultat de cette suppression :* $<$ (20) (40) (50 60) $>$. *Les séquences maximales, obtenues à l'étape $j = 4$ sont énumérées à la figure 6.6.*

Considérons à présent la prise en compte, par l'algorithme Ise, de l'ajout de nouveaux clients, au travers de l'exemple suivant.

freqInc	$freqSeed$	$freqExt$
< (10) (50 60) (80) >	< (10 20) (30) (50) >	< (60) (90) >
< (20) (50 60) (80) >	< (10 20) (40) (50) >	
< (30) (50 60) (80) >	< (10 20) (30) (60) >	
< (40) (50 60) (80) >	< (10 20) (40) (60) >	
< (20) (30) (50 60) >	< (10 20) (30) (80) >	
< (20) (40) (50 60) >	< (10 20) (40) (80) >	
< (10 20) (50 60) >		
< (10) (30) (50 60) >		
< (10) (40) (50 60) >		
< (10) (30) (50 60) >		
< (10) (30) (50) (80) >		
< (10) (40) (50) (80) >		
< (20) (30) (50) (80) >		
< (20) (40) (50) (80) >		
< (10 20) (50) (80) >		
< (10 20) (50 70) >		

FIG. 6.6 – *Les séquences maximales, obtenues à l'étape* $j = 4$

Exemple 47 *Considérons la figure 6.2, illustrant l'ajout de nouvelles transactions ainsi que de nouveaux clients. En analysant db, nous pouvons évaluer le support de chaque item sur l'incrément:* {(< (10 > ,1),(< (40) > ,1),(< (50) > ,2),(< (60) > ,2),(< (70) > ,2),(< (80) > ,3),(< (90) > ,1),(< (100),1) >}. *En combinant ces résultats avec ceux de* L_1^{DB}, *nous obtenons* L_1^{db} = {< (10) > , < (40) > , < (50) > , < (60) > , < (70) > , < (80) >}. *Comme un client a été ajouté, une séquence, pour être fréquente, doit être supportée par au moins trois transactions. Considérons à présent* L^{DB}. *L'ensemble* L_1^{DB} *devient:* {(< (10),4 >),(< (20) > ,3),(< (40) > ,3). *Ce qui signifie que l'item* 30 *est supprimé de* L_1^{DB} *puisqu'il n'est plus fréquent.*
D'après la propriété 4 (page 162), nous pouvons supprimer de L_2^{DB} *les séquences contenant cet item, ce qui revient à écrire que* L_2^{DB} = {(< (10 20),3 >) *et que l'ensemble* L_3^{DB} *est totalement supprimé. Nous pouvons désormais générer, à partir de* L_1^{db}, *de nouveaux candidats dans* 2-*candExt:*

Items	sous-sequences fréquentes
10	$< (10) >_0$ $< (20) >_0$ $< (10\ 20) >_0$
40	$< (10) >_2$ $< (20) >_2$ $< (10\ 20) >_2$
50	$< (10) >_3$ $< (20) >_3$ $< (10\ 20) >_3$
60	$< (10) >_2$ $< (20) >_2$ $< (10\ 20) >_2$
70	$< (10) >_2$ $< (20) >_2$ $< (10\ 20) >_2$
80	$< (10) >_2$ $< (20) >_2$ $< (10\ 20) >_2$

FIG. 6.7 – *Les sous-séquences fréquentes apparaîssant avant les items de* L_1^{db}

$\{<(10\ 40)>,\ <(10)\ (40)>,\ <(10\ 50)>,\ldots\ <(70)\ (80)>\}$. *En faisant une passe sur db, nous ne gardons de ces candidats que ceux qui apparaissent au moins une fois sur db. Ensuite, grâce à une analyse de U, nous pouvons déterminer les séquences candidates de taille 2 qui deviennent fréquentes et aussi quelles séquences du nouveau* L^{DB} *précèdent des séquences de* L_1^{db}. *Trois séquence candidates seulement vérifient le support minimum :* $2\text{-}freqExt = \{<(10)\ (70)>,\ <(10)\ (80)><(40)\ (80)\}$. *Observons à présent, avec plus d'attention et grâce au tableau de la figure 6.7, les sous-séquences fréquentes apparaissant avant les items de* L_1^{db} *. Le support minimum n'est alors respecté que pour les séquences précédant l'item* 50. *Nous avons alors :* $freqSeed = \{<(10\ 20)\ (50)>\}$. *Dans la mesure où aucun candidat ne peut-être généré à partir de* $freqSeed$ *et* $2\text{-}freqExt$, *le processus est fini et toutes les séquences fréquentes maximales sont ajoutées dans* $L^U = \{<(10\ 20)\ (50)>,\ <(10)\ (70)>,\ <(10)\ (80)>,\ <(40)\ (80)>,\ <(60)>\}$.

6.2 L'algorithme ISE

L'algorithme ISE, dont le principe repose sur les explications données jusqu'ici, est décrit à la figure 6.8.

Pour prouver qu'ISE construit l'ensemble des séquences fréquentes sur U, nous prouvons d'abord, dans les deux lemmes suivants, que tout nouveau fréquent peut être décomposé en deux sous-séquences. La première est un fréquent de DB et la seconde apparaît au moins une fois sur *db*.

Lemme 1 *Soit F une séquence fréquente sur U telle que* $F \notin L^{DB}$. *Alors le dernier itemset de F apparaît au moins une fois dans db.*

Preuve:

- si $|F| = 1$: comme $F \notin L^{DB}$, F contient un itemset qui apparaît au moins une fois dans *db*, donc F se termine avec un itemset qui apparaît au moins une fois dans *db*.
- si $|F| > 1$: soit S une séquence fréquente sur U, alors S=$<<A><B>>$ avec A et B des séquences telles que $0 \leq |A| < |F|$, $0 < |B| \leq |F|$, $|A| + |B| = |F|$ et $B \notin db$. Soit M_B l'ensemble des séquences qui contiennent B. Soit M_{AB} l'ensemble des séquences qui contiennent F. Nous savons que si $|M_B| = n$ et $|M_{AB}| = m$ alors $\sigma \leq m \leq n$. De plus $M_{AB} \in DB$ (car $B \notin db$ et les transactions sont ordonnées dans le temps) donc $<<A><B>>$ est fréquent sur DB et $S \in L^{DB}$. Donc si une séquence fréquente F sur U ne se termine pas par un itemset de *db*, alors $F \in L^{DB}$ et, par opposition, si F n'apparait pas dans L^{DB}, F se termine par un itemset qui apparait au moins une fois dans *db*. □

Lemme 2 *Soit* $F = <<D><S>>$, *avec D et S deux séquences telles que* $|D| \geq 0$, $|S| \geq 1$, *une séquence fréquente sur U telle que* $F \notin L^{DB}$, *alors S apparaît au moins une fois dans db et D est incluse dans (ou est) une séquence de* L^{DB}.

Preuve:

- si $|S| = |F|$: alors $|D| = 0$ et $D \in L^{DB}$.
- si $1 \leq |S| < |F|$: alors $D =< (i_1)(i_2)..(i_{j-1}) >$ et $S =< (i_j)..(i_t) >$ avec S la sous-séquence maximale qui termine F et qui apparaît au moins une fois dans *db* (grâce au lemme 1 nous savons

Algorithm ISE
Input: DB la base de données d'origine, L^{DB} l'ensemble des séquences fréquentes sur DB, l'incrément db et $minSupp$ le support minimum.
Output: L^U l'ensemble des séquences fréquentes sur $U = DB \cup db$
Method:

//première phase sur db
$L_1^{db} \leftarrow \emptyset$
foreach $i \in db$ **do**
if $(support_{DB \bigcup db}(i) \geq minSupp)$ **then** $L_1^{db} \leftarrow L_1^{db} \bigcup \{i\}$;
enddo
2-$candExt \leftarrow L_1^{db} \times L_1^{db}$
Eliminer de 2-$candExt$ les séquences $s/s \notin db$
générer l'ensemble des sous-séquences de L^{DB};
passe sur U : valider les 2-$candExt$ et construire $freqSeed$;
2-$freqExt \leftarrow$ frequent sequences from 2-candExt;

// Phases suivantes
j=2
While (j-$freqExt$!=$\emptyset$) **do**
$candInc \leftarrow$ générer les candidats depuis $freqSeed$ et j-$freqExt$;
j++;
j-$candExt \leftarrow$ générer les candidats depuis j-$freqExt$;
Filtrer les séquences de j-$candExt$ $s/s \notin db$;
if (j-$candExt$!=$\emptyset$ OR $candInc$!=$\emptyset$) **then**
Valider j-$candExt$ et $candInc$ sur U
endif
Mettre à jour $freqInc$ et j-$freqExt$;
enddo
$L^U \leftarrow L^{DB} \bigcup$ {séquences maximales de $freqSeed \bigcup freqInc \bigcup freqExt$};
end Algorithm ISE

FIG. 6.8 – *L'algorithme* ISE

que $|S| \geq 1$). Soit M l'ensemble de toutes les séquences qui contiennent F et $time_u$ la date de mise à jour. $\forall s \in M$, $s =< < A > < B > >$ / $\forall i \in A$, $time_i < time_u$, $\forall j \in B$, $time_j \geq time_u$, nous avons $< (i_{j-1})(i_j)..(i_t) > \not\subset B$ (car S est la sous-séquence maximale qui termine F et qui apparaît au moins une fois dans db). Alors, comme $\forall i$, M_i (la i^{eme} séquence de M) contient F, et comme les transactions dans la base de données sont ordonnées dans le temps, $< (i_1)(i_2)..(i_{j-1}) > \subset A$. Donc $\forall s \in M$, $\exists A$ une sous-séquence qui commence s telle que $A \in DB$, $D \subseteq A$. Donc $D \in L^{DB}$ □

Considérons les deux sous-séquences d'une nouvelle séquence fréquente. Nous prouvons grâce au lemme suivant, que la deuxième est générée en tant qu'extension candidate par ISE.

Lemme 3 *Soit F une séquence fréquente sur U telle que F n'apparaît pas sur L^{DB}. F être écrite $< < D > < S > >$ avec D et S deux séquences telles que $|D| \geq 0$, $|S| \geq 1$, S apparaît au moins une fois dans db, D est incluse dans (ou est) une séquence de L^{DB} et $S \in candExt$.*

Preuve: Grâce au Lemme 2, nous devons seulement prouver que $S \in candExt$.

- si S ne contient qu'une transaction, réduite à un item : S est donc trouvée dans la première passe sur db et ajoutée à 1-$candExt$.
- si S contient plus d'un item : $candExt$ est construit suivant la méthode GSP à partir des extensions fréquentes apparaissant sur db. $candExt$ est donc un surensemble des séquences fréquentes sur U qui apparaîssent sur db. □

Le théorème suivant garantit la validité de l'algorithme ISE.

Théorème 6 *Soit F une séquence fréquente sur U telle que $F \notin L^{DB}$. Alors F est générée comme séquence candidate par* ISE.

Preuve: A partir du Lemme 2 considérons les différentes déclinaisons possibles de S.

- si $S = F$: alors S sera générée dans $candExt$ (Lemme 3) et ajoutée à $freqExt$.
- si $S \neq F$:
  - si S ne contient qu'une transaction, réduite à un item : alors l'association $< < D > < (i) > >$ sera ajoutée dans $freqSeed$.
  - si S contient plus d'un item : considérons i_{1_1} le premier item du premier itemset de S. i_{1_1} est fréquent sur db, donc $< < D > < (i_{1_1}) > >$ sera généré dans $freqSeed$. D'après le Lemme 3, S apparaît dans $freqExt$ et sera utilisée par ISE pour construire $< < D > < S > >$ dans $candInc$ qui sera finalement ajouté à $freqInc$ □

6.3 Optimisations

Dans le but d'améliorer les temps de réponse offerts par l'algorithme ISE, nous avons mis en place deux techniques d'optimisation relatives à la génération des candidats. Cette volonté de réduire le nombre de candidats générés vient du simple constat que tous les algorithmes de fouille de données, basés sur des méthodes du type générer-élaguer, voient leur performances ralenties par un trop grand nombre de candidats à tester et cherchent donc à réduire ce nombre au maximum.

La première technique que nous avons mise en place, utilise les informations que l'on peut obtenir de l'ensemble L^{db}, i.e. les séquences fréquentes sur *db*. Cette optimisation est basée sur le lemme suivant :

Lemme 4 *Soient $s \in freqSeed$ et $s' \in freqExt$, deux séquences découvertes lors de l'application de l'algorithme* Ise, *telles que le dernier item i de s est aussi le premier item de s'. S'il existe un item $j \in L_1^{db}$ tel que $j \in s'$ et j n'est pas précédé par s, alors le candidat issu de la génération à partir de s et s' ne sera pas fréquent.*

Preuve: Si s n'est pas suivi par j, alors $< s\ j >$ n'est pas fréquent. Donc $c =< s\ s' >$ n'est pas fréquente, car il existe une sous-séquence de c qui n'est pas fréquente.

L'utilisation de ce lemme, lors de la génération des candidats, permet de réduire le nombre de candidats générés de manière significative. Nos expérimentations ont montré un gain d'environ 40% sur le nombre de candidats générés. De plus, le seul coût engendré par cette optimisation est celui d'une recherche de chaque item de la séquence s' dans le tableau des précédences de la sous-séquence s (le coût de cette recherche est donc quasiment inexistant).

L'exemple suivant illustre le comportement de cette optimisation.

Exemple 48 *Considérons $s =<$ (50 70) $>$ une des séquences fréquentes de 2-$freqExt$. De plus nous savons que $<$ (10) (30) (50) $>$ fait partie des séquences fréquentes de $freqSeed$. Selon la description de la phase de génération, nous devrions créer le candidat $<$ (10) (30) (50 70) $>$. Cependant la séquence $<$ (10) (30) $>$ n'est pas suivie par* 70. *Nous pouvons donc en conclure que $<$ (10) (30) (70) $>$ n'est pas une séquence fréquente. Or cette dernière est une sous-séquence de $<$ (10) (30) (50 70) $>$, donc avant de la générer nous savons qu'elle ne sera pas déterminée fréquente. Cette séquence candidate n'est donc pas générée.*

L'idée principale de la seconde optimisation consiste à éviter la génération de candidats, si l'on connait déjà leur existence en tant que séquence fréquente grâce à une étape précédente. En effet, quand nous générons un candidat en concaténant deux séquences de $freqExt$ et $freqSeed$, nous cherchons d'abord la présence de ce candidat dans $freqSeed$ et $freqInc$. Si cette présence est vérifiée alors le candidat est supprimé. Pour illustrer cette optimisation, considérons que $<$ (30) (40) $>$ appartient à 2-$freqExt$. Considérons à présent que $<$ (10 20) (30) (40) $>$ et $<$ (10 20) (30) $>$ appartiennent à $freqSeed$. La génération de candidats devrait aboutir à la séquence $<$ (10 20) (30) (40) $>$. Or cette séquence a déjà été découverte comme séquence fréquente. Cette optimisation a permis de réduire d'environ 5% (toujours d'après nos expérimentations) le nombre de candidats à tester, pour un coût extrêmement faible.

6.4 Experimentations

Cette section a pour but de présenter les expérimentations réalisées afin d'évaluer les performances de l'algorithme Ise, notamment en comparaison de GSP. Ces expérimentations ont été réalisées sur une machine de type Ultra Sparc (Enterprise 2) équipée d'un processeur cadencé à 200 MHz, avec 256 Mo

|D|	Nombre de séquences (taille de la base)
|C|	Nombre moyen de transactions par client
|T|	Nombre moyen d'items par transaction
|S|	Longueur moyenne des fréquents de taille maximale
|I|	Longueur moyenne des itemsets dans les fréquents de taille maximale
N_S	Nombre de fréquents de taille maximale
N_I	Nombre d'itemsets fréquents de taille maximale
N	Nombre d'items
I^-	Nombre d'itemsets supprimés de U pour construire *db*
$D^{\%}$	Pourcentage de transactions mises à jour dans U

TAB. 6.2 – *Paramètres*

de mémoire vive, faisant fonctionner un système Unix System V Release 4 et sur un disque dur SCSI distant de 9 Go.

6.4.1 Jeux de données

Les expériences ont été réalisées sur des jeux de données synthétiques dont les caractéristiques sont décrites dans les tableaux 6.2 et 6.3.

Nous avons utilisé un générateur de données synthétiques, proposé par [SA96b] [1]. Ensuite la création de DB et *db* se déroule de la manière suivante. Pour respecter le plus possible un modèle réaliste, comme dans [CHNW96], nous avons tout d'abord généré toutes les transactions, correspondants à la base de données U, puis déterminé la frontière entre DB et *db* à l'intérieur de cette base, avant de la décomposer en DB+*db*.
Dans le but de déterminer la capacité d'ISE à extraire les motifs séquentiels de façon incrémentale, nous avons supprimé de la base U des itemsets selon le paramètre I^- spécifié par l'utilisateur. Le nombre de transactions ayant subit cette suppression étant, quant à lui, obtenu via le paramètre $D^{\%}$. Grâce à ce paramètre, $D^{\%}$ des transactions sont choisies afin de leur retirer I^- items.

Enfin, les transactions supprimées sont stockées dans la base de données incrément *db* alors que la base générée (après suppression des transactions déplacées vers *db*) est renommée DB. Pour retrouver U, la base de données d'origine après ces modifications, il suffit de concaténer DB et *db*. Un dernier paramètre de ces manipulations, destiné à tester le comportement d'ISE en cas de suppression de clients, est appelé $C^{\%}$. Il permet de déterminer le nombre de clients à déplacer de la base générée vers *db*.

La table 6.2 donne la liste des paramètres utilisés lors de la génération des bases de données de test, alors que la table 6.3 résume les caractéristiques des bases que nous avons générées. Nos premières expérimentations ont été réalisées sur des bases où l'incrément ne contient que des nouvelles transactions. Pour ce cas, I^- vaut 4 et $D^{\%}$ vaut 90%. Enfin, les expérimentations réalisées avec de nouveaux clients concernent une suppression de clients, $C^{\%}$, de 10% et 5%.

1. Le générateur peut être trouvé à l'adresse suivante: (*http://www.almaden.ibm.com/cs/quest).*

Nom	|C|	|I|	N	|D|
C9-I4-N2K-D100K	9	4	2,000	100,000
C13-I3-N20K-D500K	13	3	20,000	500,000
C15-I4-N30K-D600K	15	4	30,000	600,000
C20-I4-N2K-D800K	20	4	2,000	800,000

TAB. 6.3 – *Paramètres des jeux de données utilisés*

6.4.2 Comparaison entre ISE et GSP

Nous comparons, dans cette section, l'approche incrémentale et une approche naïve qui consiste à utiliser GSP sur U sans tenir compte de L^{DB}. Nous avons également testé la stabilité d'ISE en fonction de la taille des bases de données traitées. Finalement, nous avons mené une étude destinée à analyser le comportement d'ISE en fonction de l'importance de la mise à jour.

Comparaison d'ISE avec l'approche naïve

La figure 6.9 exprime les résultats obtenus sur différents jeux de données, avec un support minimum ajusté pour obtenir les temps de réponses les plus significatifs (i.e. le plus bas possible tout en gardant des temps de réponse raisonnables). Les temps de réponse d'ISE se retrouvent sous le nom "Incremental Mining", alors que ceux de GSP sont identifiés par "GSP". La désignation "Mining from scratch" sera expliquée dans la section 6.5.

Le figure 6.9 met clairement en valeur la différence de performance entre les deux algorithmes et l'on peut y observer que cette différence augmente quand le support diminue. Nous pouvons y observer qu'ISE est entre 3.5 et 4 fois plus rapide que GSP utilisé sur U. Nous pouvons également y remarquer qu'ISE surpasse GSP pour les petits supports comme les plus élevés : ISE est toujours 2.5 à 3 fois plus rapide pour les supports élevés et ce, même si le nombre d'itemsets est grand. Par exemple le dernier graphique de la figure 6.9 illustre une expérimentation réalisée dans le but d'analyser les effet d'un nombre d'itemsets sur la performance des algorithmes mis en jeu. Quand le support baisse, l'algorithme GSP offre les performances les plus faibles.

Cela s'explique par le fait que le nombre de candidats générés pour obtenir les séquences fréquentes dans ISE est plus faible. La méthode de génération mise en place, basée sur les techniques présentées plus haut, utilise les informations collectées dans l'incrément. En effet, après un parcours de *db* nous savons quelles séquences peuvent être fréquentes de façon plus précise que dans GSP. De plus, ce nombre est considérablement réduit par la prédiction de "non fréquence" parmi les candidats. Nous étudierons plus précisément les effets de ces optimisations dans la section 6.5

Effet du nombre de données traitées en entrée

Nous avons étudié le comportement d'ISE dans le cas on le nombre de transactions des bases de données à traiter augmente. Le résultat attendu étant un comportement linéaire de la part de notre algorithme. Ce résultat est effectivement celui que nous avons obtenu, comme le confirme la figure 6.10, où nous pouvons observer qu'ISE adopte des temps de réponse linéaires alors que la taille des bases de données

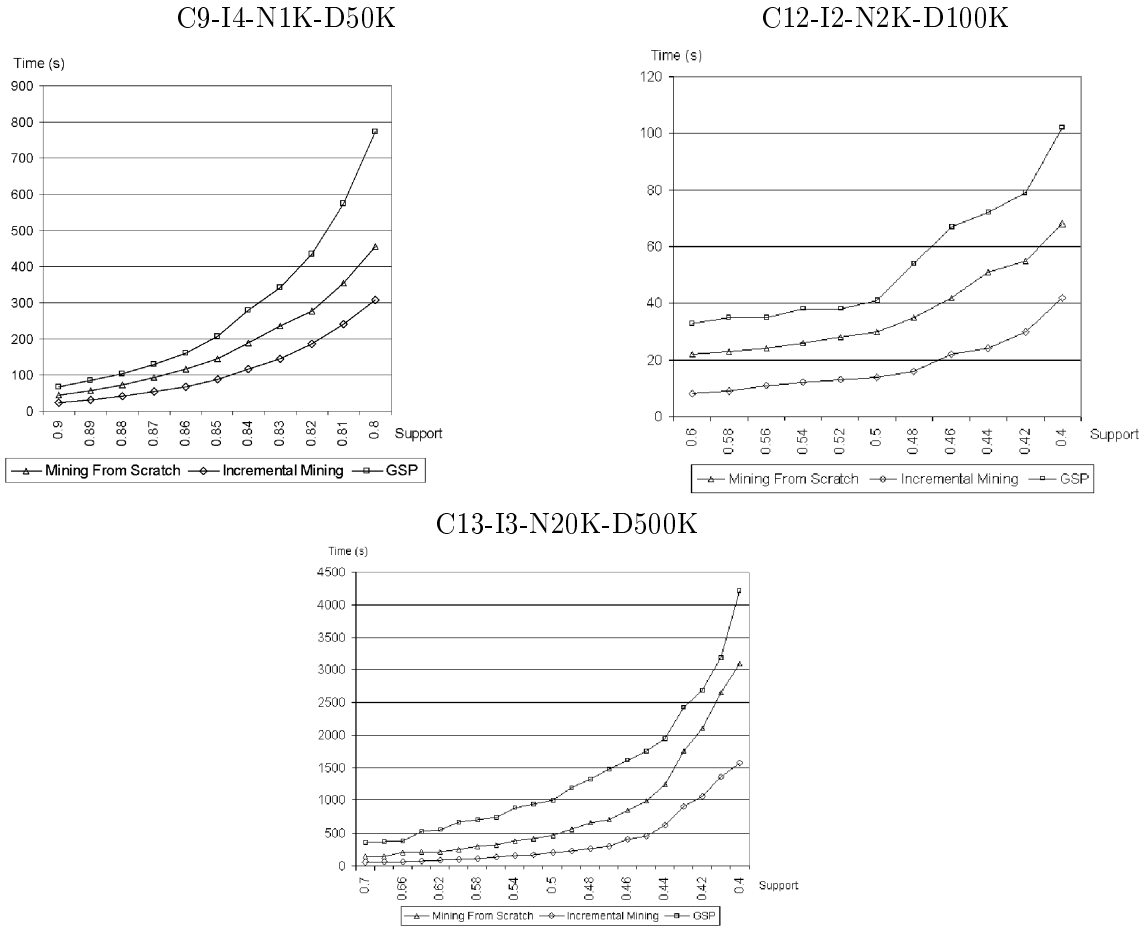

FIG. 6.9 – *Execution times*

traitées est multiplié par un facteur allant jusqu'à 10. Ces expérimentations ont été menées sur le jeux de données C12-I4-N1K-D50K et pour réaliser cette évaluation de la façon la plus complète, le support minimum a été fixé sur trois valeurs différentes (2%, 1.5% et 1%). A cours de cette évaluation, la taille de l'incrément est restée proportionnelle ($D^{\%} = 90\%$ et $I^{-} = 4$) au nombre de transaction dans la base traitée.

Effets de la taille de la mise à jour

Les expérimentations que nous avons menées, et qui sont présentées dans cette section, sont destinées à analyser les performances d'ISE en fonction de la taille de la mise à jour. Ces expérimentations ont été menées sur les bases C13-I3-N20K-D500K et C12-I2-N2K-D100K avec des support respectifs de 0,6% et 0,4%. A partir de ces bases, nous avons fait varier le nombre de transactions supprimées (taille de l'incrément) ainsi que le nombre de clients. Les transactions effacées sont stockées dans la base *db* tandis que les transactions restantes constituent la base DB. Nous avons tout d'abord procédé à un

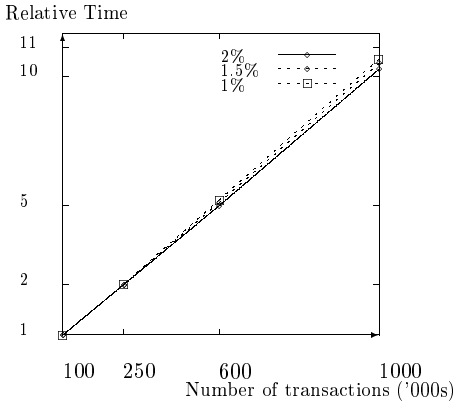

FIG. 6.10 – *Nombre total de transactions*

traitement par GSP pour obtenir L^{DB} avant d'utiliser ISE sur la base de données mise à jour, U. La figure 6.11 montre le résultat de ces expérimentations après les relevés de temps d'exécution affichés par ISE.

En ce qui concerne la première expérimentation, nous pouvons constater qu'ISE offre des temps de réponse efficaces, pour un nombre d'itemsets supprimés allant de 1 à 6. Comme nous pouvions le prévoir, le temps requis par ISE pour déterminer L^U augmente avec le nombre de transactions supprimées de la base d'origine. Par exemple, quand 10 itemsets sont supprimés de la base d'origine, ISE requiert 180 secondes pour 30% de la base touchée par la mise à jour, contre 215 secondes quand cette mise à jour concerne toutes les entrées de la base. Ce phénomène s'explique principalement par le fait que le nombre de changements dans la base de données est tel, que les connaissances acquises précédemment sont inutiles. En revanche, il est intéressant de noter que le comportement d'ISE, ne semble pas dépendre du nombre de transactions affectées par la mise à jour.

Considérons la seconde surface. L'algorithme demande un temps de plus en plus important, en fonction du nombre d'itemsets supprimés de la base. Néanmoins, lorsque 3 itemsets sont supprimés de la base générée, ISE ne demande que 30 secondes avant de découvrir l'ensemble des motifs séquentiels.

Impact du nombre de clients ajoutés

Nous proposons de limiter notre étude, dans la mesure où cela nous semble être le plus réaliste, à l'ajout d'un nombre de clients au plus égal au nombre de clients déjà enregistrés dans la base d'origine. Une première intuition pourrait nous conduire à étudier le comportement d'ISE en faisant varier le nombre de clients ajoutés de manière croissante. En fait cette approche naïve ne peut-être un indicateur fiable. En effet, comme l'illustre la figure 6.12 (dans laquelle les parties grisées ne contiennent aucunes données), quand le nombre de clients ajoutés (la taille de *db*) augmente, le support minimum (en nombre de clients minimum) augmente également. Cela est du au fait que la taille de la base augmente d'autant de clients que *db* en apporte. De ce fait, le nombre de fréquents découverts précédemment diminue car

C13-I3-N20K-D500K C12-I2-N2K-D100K

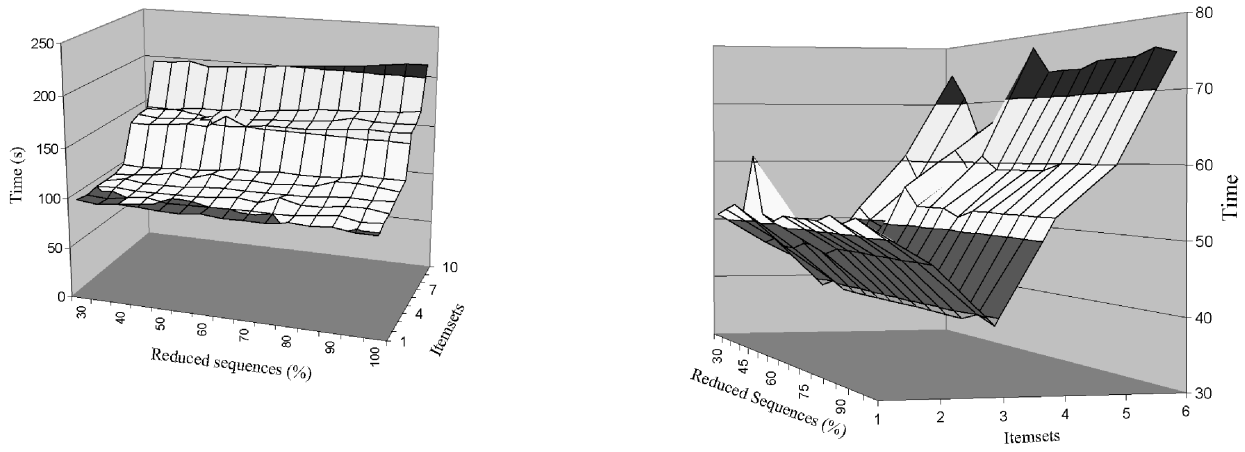

FIG. 6.11 – *Taille de la mise à jour*

les séquences qui étaient fréquentes dans L^{DB} voient leur support chuter. Il en résulte également une augmentation du nombre de nouveaux fréquents, ce qui revient à dire que la méthode incrémentale se retrouve :

- en position de faiblesse si le nombre d'itemsets ajoutés est grand et riche en fréquents.
- en position de trop grande facilité si le nombre d'itemsets est faible, car aucun nouveau fréquent ne sera plus découvert.

Une approche bien plus intéressante, destinée à évaluer les performances d'ISE dans le cas de l'ajout de nouveaux clients, consiste à comparer les temps d'exécution offert par cet algorithme, avec ceux de GSP dans le cas d'un ajout de client fixé, pour différentes valeurs du support. la figure 6.13 illustre les résultats obtenus pour les jeux de données C9-I4-N2K-D100K et C20-I4-N2K-D800, avec un nombre de clients supprimés respectivement fixé à 10% puis 5%. Nous pouvons y observer qu'ISE est très efficace et qu'en cas d'ajout de client il permet d'obtenir les résultats en deux fois moins de temps qu'une exécution de GSP sur la base U.

6.5 Discussion : ISE sorti du contexte incrémental

Au cours de nos expérimentations, nous avons pu constater l'efficacité d'ISE pour l'extraction incrémentale de motifs séquentiels. Pour évaluer cette efficacité, nous avons mesuré les temps de réponse d'ISE sur U, d'un côté, par opposition à ceux de GSP, de l'autre. Nous avons noté un facteur qui nous a semblé particulièrement intéressant. Il s'agit du cumul des temps nécessaires pour:

- découper U en $DB + db$,
- utiliser GSP sur DB,
- utiliser ISE sur db.

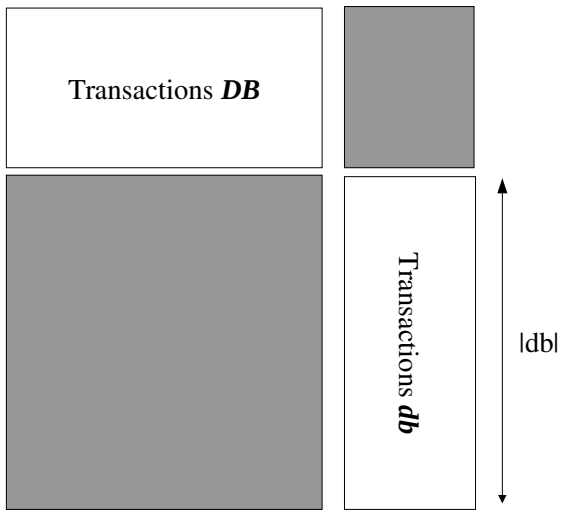

FIG. 6.12 – *effets d'une augmentation du nombre de clients*

Ce cumul de temps (qui n'était pas destiné à être chronométré dans l'optique originale de ces expérimentations) était parfois plus faible que le temps nécessaire à GSP pour fonctionner sur U. Afin d'explorer les cas pour lesquel ce phénomène pouvait se produire, nous avons mené les expérimentation décrites dans cette section. Nous nous intéressons donc, dans cette section, aux capacités de l'algorithme ISE à extraire des motifs séquentiels sur U à partir d'une information nulle.

Les jeux de données utilisés dans cette partie des expérimentations sont les mêmes que dans la section 6.4.2. Les opérations que nous avons réalisées sur chaque jeu de données sont les suivantes. Tout d'abord, retirer 6 items de 60% des transactions de la base dans le but de créer un incrément. Ensuite, nous avons utilisé GSP pour extraire les sequences fréquentes de taille k dans DB. Enfin, nous avons utilisé ISE pour connaitre les fréquents sur U. le temps d'exécution cumulé de ces trois étapes est alors reporté à la figure 6.9 sous la désignation "Mining from scratch". en comparaison d'une phase de fouille de données correspondant à GSP sur cette même base de données, le temps cumulé des trois opération sur U peut alors s'avérer 1,7 à 3 fois plus rapide que celui de GSP sur cette même base (U). Ce gain de performance s'explique principalement par la réduction du nombre de candidats générés. Nous étudions les effets de cette réduction de candidats dans la section suivante.

Nombre de candidats générés

Dans le but d'expliquer la corrélation entre le nombre de candidats et les temps d'exécution, nous avons comparé le nombre de candidats générés par GSP et par ISE. Les résultats sont décrits par la figure 6.14. Comme nous pouvons le constater, le nombre de candidats de GSP est approximativement le double du nombre de candidats générés par ISE. Observons ce phénomène de façon plus précise pour un support faible. Dans la première série de courbes, GSP génère plus de 7000 candidats, quand ISE

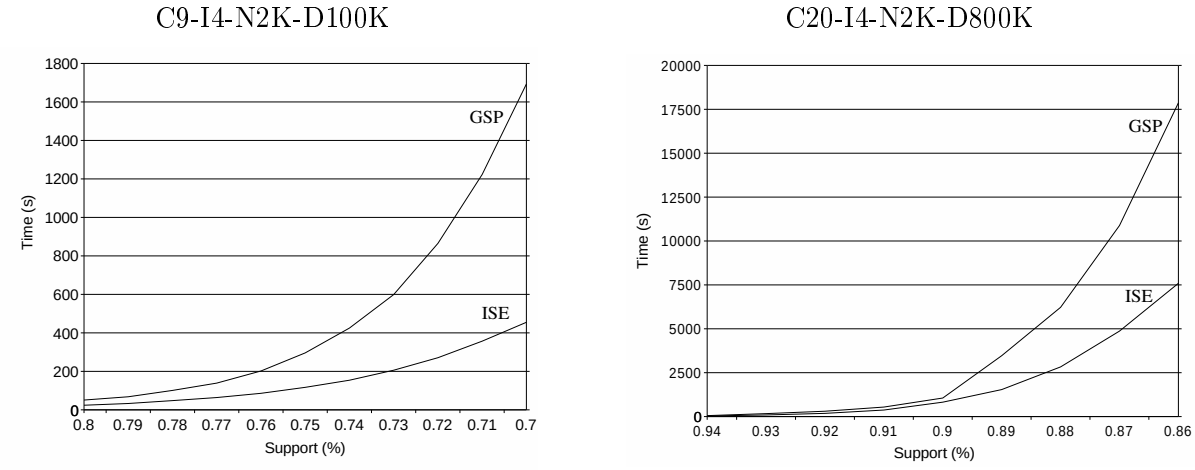

FIG. 6.13 – *Temps d'exécution avec un nombre de clients supprimés fixé à 10% et 5%*

n'en génère que 4000. Le même phénomène peut être observé dans la deuxième série de courbes, où l'on observe que GSP génère plus de 14000 candidats, alors qu'ISE en génère 8000.

Nous pouvons également observer sur cette figure une mise en corrélation qui montre de façon directe les liens existants entre le nombre de candidats générés et les temps d'exécution.

Variations de l'importance de la mise à jour

Cette dernière batterie d'expérimentation a pour objectif d'analyser les performances d'ISE en fonction de la taille de la mise à jour. Les résultats ont été obtenus à partir des fichiers C12-I2-N2K-D100K et C13-I3-N20K-D500K avec un support de 0,4%.

Nous avons procédé de la façon suivante : pour chaque couple (i,j), avec $i \in [i_a..i_z]$, $j \in [j_A..j_Z]$, a le nombre minimal d'itemsets supprimés, z le nombre maximal d'itemsets supprimés, A le pourcentage minimal de transactions affectées et Z le pourcentage maximal de transactions affectées, nous avons relevé le temps d'exécution correspondant au cumul $TempsSuppression_{(i,j)} + Temps_{GSP} + Temps_{ISE}$. Les résultats de ces relevés se présentent donc sous la forme de surfaces, reportées à la figure 6.15.

Considérons la première surface de la figure 6.15. Le meilleur résultat est obtenu quand 5 itemsets sont supprimés de la base U. Tous les fréquents sont alors obtenus en moins de 17 secondes. L'algorithme garde son efficacité de 2 jusqu'à 7 itemsets supprimés mais commence à donner des signes de faiblesse à partir de 5 itemsets retirés de 10% des clients. Dans la seconde surface de la figure 6.15, les performances d'ISE sont comparables et les meilleurs résultats sont obtenus pour 9 itemsets, retirés de 80% des clients.

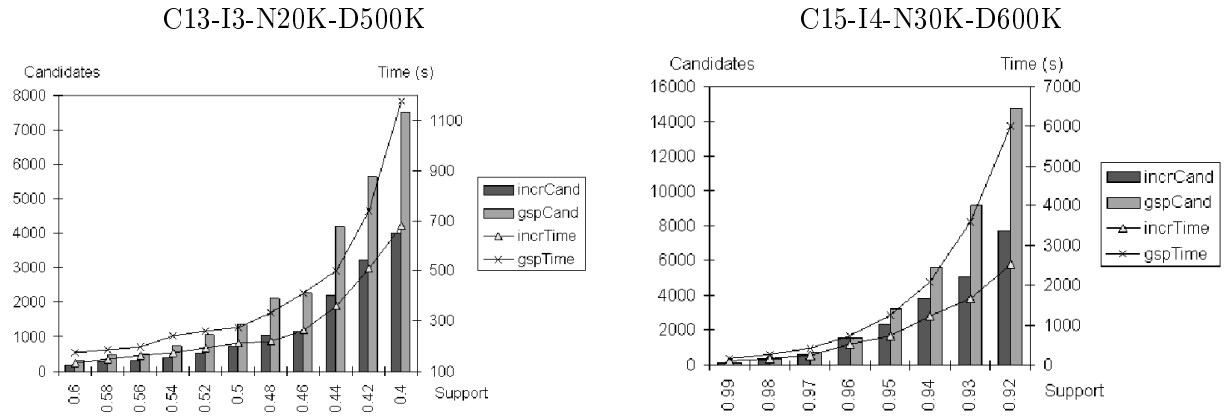

FIG. 6.14 – *Nombre de candidats*

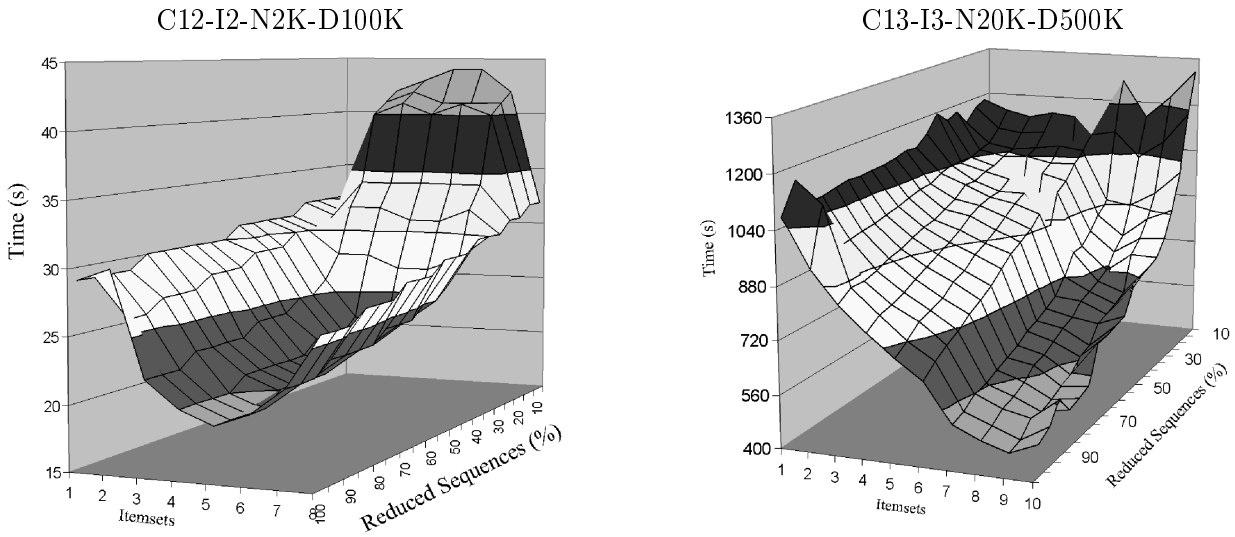

FIG. 6.15 – *Taille de la mise à jour*

Chapitre 7

Applications

La recherche de motifs séquentiels, par sa nature, se prête à de nombreuses adaptations. Parmi ces applications, certaines peuvent sembler directes (comme le Web Usage Mining que nous aborderons et améliorerons dans la partie III) et d'autres peuvent demander un prétraitement particulier afin de les rendre compréhensibles par un algorithme d'extraction de motifs séquentiels. Ce prétraitement, qui est à la base du bon fonctionnement de l'application, consiste à acquérir les données en entrée et à les transformer de façon pertinente pour l'algorithme de fouille de données. Comme nous allons le constater, c'est de cette transformation que dépend la qualité et le sens des résultats obtenus par le processus de fouille de données. La section 7.1 explique comment un ensemble de textes peut être exploité par un processus de recherche de motifs séquentiels, dans le but de trouver des structures de texte (articulations de sous-phrases) communes au textes de cet ensemble. La section 7.2, quant à elle, montre que la transformation des données peut connaître un aspect moins trivial, pour des problématiques qui pourraient sembler éloignées de celle de la recherche des motifs séquentiels.

7.1 Text Mining

Parmi les applications que l'on peut trouver pour un algorithme de fouille de données et plus particulièrement un algorithme d'extraction de motifs séquentiels, une des plus intéressantes réside certainement dans l'analyse de fichiers textes. Si l'on constitue un ensemble de textes qui présentent un facteur commun (cela peut être leur auteur, le ou les thèmes qu'ils abordent, mais aussi leur source et bien d'autres...) et que l'on dispose ces textes selon la même organisation que des enregistrements dans une base données, il est possible d'extraire, de cet ensemble de textes, des motifs séquentiels dont l'intérêt est évident. Par analogie avec l'exemple du supermarché, cela revient à considérer les textes comme des clients, les phrases commes des dates et les mots comme des items. De plus, une utilisation pertinente de la temporalité du problème peut être faite en considérant les paragraphes pour éloigner des phrases dans le temps (et donc tenir compte de leur distance de séparation dans le texte).

Une utilisation des motifs séquentiels sur des fichiers texte est présentée dans [LAS97]. Cette approche se place dans l'axe de la découverte de tendances dans des textes qui subissent une évolution. Le but recherché permet de découvrir, en explorant la base de données des dépots de brevets accordés aux États Unis, les travaux de recherche récents de différentes sociétés. Pour cela, les textes sont groupés par année et la base est explorée en ne considérant que les textes d'une année. Il est alors possible de constater l'évolution des intérêts portés aux différents sujets pour les dépots de brevets, d'année en année.
Notre approche se distingue de celle proposée par [LAS97], par la façon dont les phrases sont interprétées. Nous définissons le problème de la recherche de tendances dans un ensemble de textes comme ceci :

- Un mot est noté w.
- Une phrase est un ensemble de n mots $(w_1 w_2 \dots w_n)$.
- Un paragraphe P est une liste ordonnée de phrases noté $< p_1 p_2 \dots p_k >$, avec p_i les phrases de ce paragraphe.
- Un texte T est le résultat de la juxtaposition des différents paragraphes de ce texte. Il est noté de la même façon qu'un paragraphe. Des phrases appartenant à un même paragraphe sont

séparées par une durée moins importante que des phrases appartenant à des paragraphes différents ($\forall p_i,p_j \in P, \forall p_k \in P_k, \forall p_l \in P_l/P_k \neq P_l, dist(p_i,p_j) < dist(p_k,p_l)$). La séquence représentant le texte est trié par date des paragraphes croissante (l'estampille des paragraphes croît avec leur ordre d'apparition dans le texte).

– L'ensemble des séquences que constituent tous les textes transformés, représente alors une instance pour le problème de la recherche de régularité.

De plus nous appliquons une procédure plus impliquée dans le domaine de l'ECD, dans la mesure où nous devons filtrer les textes avant de les transcrire en données lisibles pour la fouille de données. Il est, en effet, inutile de prendre en compte les mots de liaison utilisés dans les phrases ou d'autres expressions que la fouille de données trouverait forcément (et vainement) fréquentes.

Nous avons appliqué cette méthode sur des textes de différentes sources trouvées sur internet : les discours du président de l'assemblée nationale [1] et l'éditorial de Sciences&Vie [2].

Parmi les répétions détectées, nous pouvons en citer deux concernant les discours à l'assemblée nationale :

{ <(c'est) (plaisir)(je)>, <(université) (chercheurs)> }

Alors que la première n'offre pas beaucoup d'intérêt et devrait être supprimée dans le processus d'ECD, la seconde est plus significative d'une tendance. De la même manière, nous avons remarqué, dans les "infos du jours de Sciences&Vie", que la régularité suivante était très importante : { <(chercheurs) (possible)> }.

7.2 Schema Mining

En définissant une composante Transactions imbriquées, nous montrons qu'à l'aide d'une transformation efficace des données initiales, nous pouvons transformer la problématique initiale en recherche de motifs séquentiels. L'approche proposée dans cette composante est décrite dans [LMP00]. Cette transformation est ensuite appliquée à la recherche de régularités sur des données semi-structurées issues du Web au travers de la spécification d'un système d'extraction dédié.

7.2.1 Composante Transactions Imbriquées

La recherche de régularités dans des transactions imbriquées fréquentes est réalisée en deux étapes.

1. *Phase de transformation* : cette étape est chargée de transformer les transactions initiales en données accessibles par la phase de recherche ;
2. *Phase de recherche* : cette étape recherche les transactions imbriquées fréquentes dans la base de données transformée.

La base de données est initialement triée avec l'identifiant de la transaction comme clé principale et les éléments simples stockés dans les ensembles d'éléments sont triés par ordre lexicographique. Cette

1. Les textes de l'assemblée nationale sont disponible sur l'URL (*http://www.assemblee-nat.fr/*).
2. L'info du jour de Sciences&Vie se trouve sur l'URL (*http://www.excelsior.fr/sv/info/*).

dernière opération nous permet de réduire considérablement l'espace de travail dans la mesure où, les éléments d'un ensemble n'étant pas ordonnés, trop de combinaisons doivent être prises en compte.
Chaque transaction contenant des éléments complexes est alors transformée de la manière suivante :

1. *Prise en compte de la profondeur d'imbrication et du type complexe* : chaque élément simple est considéré et si cet élément est une partie d'un ensemble alors l'élément simple est précédé par la lettre 'S' (respectivement la lettre 'L' s'il s'agit de liste d'éléments simples). En outre, pour prendre en compte le niveau d'imbrication de l'élément, un entier, décrivant la profondeur d'imbrication de cet élément dans la structure complexe, est ajouté à l'élément.
2. *Création de liste d'ensembles d'éléments simples* : quand deux éléments sont au même niveau et si le premier est directement suivi par le second, nous les regroupons dans un même ensemble autrement ils sont inclus dans deux ensembles séparés. La notion d'ordre de la valeur "liste-de" est, par contre, prise en compte en créant de nouveaux ensembles entre les éléments.
   La transaction "composite" qui résulte de l'union de ces ensembles obtenue à partir des transactions initiales décrit une séquence d'éléments simples modifiés et l'ordre de cette séquence peut alors être vu comme une navigation en "profondeur d'abord" des transactions.

A la fin de cette phase, la base de données DB est transformée en une nouvelle base de séquence DB' dans laquelle l'imbrication dans les transactions a été modifiée en une liste ordonnée d'éléments simples.

Exemple 49 *Pour illustrer la phase de transformation, considérons les deux transactions suivantes :* $t_1 = \{a,\{c,\{d,f\}\}\}$ *et* $t_2 = \{a, < e,\{f,b\},d, < g,h >> ,c\}$. *Dans la première transaction, étant donné que tous les éléments simples apparaissent dans un ensemble de valeurs, le symbole 'S' leur est affecté. Concernant le niveau d'imbrication de la transaction, nous affectons à chaque élément simple son niveau par rapport à l'ensemble le "plus haut" de la hiérarchie. Les éléments simples a,b,c,d et f sont alors transformés de la manière suivante :* Sa_1,Sc_2,Sd_3,Sf_3. *En affectant chaque élément simple dans un nouvel ensemble et en parcourant en profondeur d'abord la transaction, nous obtenons la transformation suivante pour la transaction* t_1 *:* $t_1 = (Sa_1)\ (Sc_2)\ (Sd_3\ Sf_3)$. *En appliquant le même principe pour la transaction* t_2*, nous obtenons :* $t_2 = (Sa_1)\ (Le_2)\ (Sf_3\ Sb_3)\ (Ld_2)\ (Lg_3)\ (Lh_3)\ (Sc_1)$. *Nous pouvons constater que la modification de la transaction* t_2 *respecte l'ordre de parcours en profondeur d'abord. Examinons en détail la partie "liste-de"* $< g : \perp,h : \perp >$ *de la transaction* t_2. *Comme les éléments g et h sont ordonnés à cause de la valeur "liste-de", nous considérons que même s'ils interviennent au même niveau, et même si h suit directement g, ils ne peuvent pas être regroupés dans un même ensemble.* □

Le fait de transformer les données initiales sous la forme de séquences d'éléments simples nous permet de nous rapprocher de la problématique de la recherche de motifs séquentiels (Chapitre 2.2, partie I). En effet, nous n'avons plus qu'à rechercher dans les données transformées quelles sont les listes d'éléments simples qui interviennent suffisamment fréquemment pour être intéressantes, i.e. quelles sont les listes d'éléments simples dont le nombre d'occurrences dans la nouvelle base de données DB' est supérieur au support minimal spécifié par l'utilisateur. Les optimisations apportées à l'algorithme de recherche de motifs séquentiels sont décrites dans [Lau00].

7.2.2 Composante Données Semi-Structurées

La problématique de la recherche de régularités dans des données semi-structurées est très similaire à celle que nous avons présentée précédemment. En fait, l'algorithme proposé permet également de résoudre ce problème. Cependant, l'analyse de tels documents pose d'autres types de problèmes qui nous ont amené à définir un système d'extraction de connaissances spécifique. Le système est décrit dans [LMPT00].

Inspiré des travaux menés dans le cadre du Web usage Mining, nous considérons que le mécanisme de la découverte de régularités dans de grandes bases de données semi-structurées est également un processus qui se fait en deux étapes. La première étape concerne la recherche des objets semi-structurés sur le Web et leur stockage dans une base. A partir de cette base, la phase de transformation modifie les données originales de la même manière que dans la composante précédente. Le résultat de cette transformation est une nouvelle base de données contenant les informations utiles sur lesquelles des techniques de fouille de données peuvent être appliquées pour rechercher les régularités dans les expressions des chemins. L'architecture fonctionnelle de notre approche est décrite dans la figure 7.1.

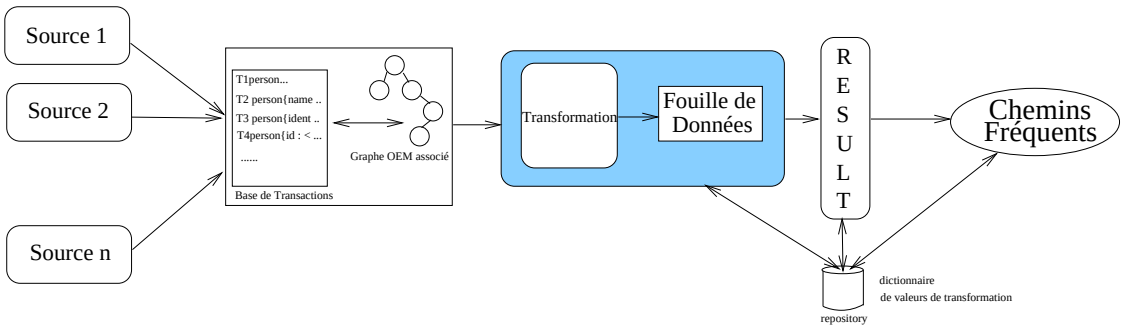

FIG. 7.1 – *Architecture générale*

Extraction des données

La structure de chaque objet de la source (page Web) doit être extraite au cours de cette phase. Tout d'abord une étape de filtrage est faite de manière à éliminer les données qui ne sont pas utiles dans l'analyse : image, vidéo, son, etc. En outre, en fonction du point de vue de l'utilisateur, les sous-structures "non intéressantes" sont également éliminées.
Cette extraction est réalisée sur un grand nombre de sources de manière à obtenir un ensemble suffisant de structures dans lesquelles nous souhaitons découvrir les sous-structures régulières.

Pour réaliser cette phase d'extraction, nous considérons que l'utilisateur final possède un parser ou un wrapper. Bien entendu, ce parser nécessite l'implémentation d'algorithmes efficaces pour extraire les structures de graphe sous-jacentes à une ou à un ensemble de pages. Par exemple, dans [HGMC+97], un outil efficace d'extraction de données semi-structurées à partir d'un ensemble de pages HTML a été proposé. La particularité de cet outil est qu'il propose directement une conversion des données extraites dans un graphe OEM.

Trans_id	expressions de chemins transformées
t_1	$(Sidentity_1)$ $(Sname_2\ Saddress_2)$
t_2	$(Sidentity_1)$ $(Sname_2\ Saddress_2)$ $(Lstreet_3)(Lzipcode_3)$
	$(Scompany_2\ Sdirector_2)$ $(Lname_3)$ $(Lfirstname_3)$
t_3	$(Sidentity_1)$ $(Sid_2\ Saddress_2)$ $(Lstreet_3)$ $(Lzipcode_3)$
t_4	$(Sidentity_1)$ $(Sname_2\ Saddress_2\ Scompany2)$
t_5	$(Sidentity_1)$ $(Sname_2\ Saddress_2)$
t_6	$(Sidentity_1)$ $(Sname_2\ Saddress_2)$ $(Lstreet_3)$ $(Lzipcode_3)$
	$(Sdirector_2)$ $(Lname_3)$ $(Lfirstname_3)$

FIG. 7.2 – *Une base de données de transactions après la phase de transformation*

Extraction de connaissances

A partir des données extraites, nous appliquons l'algorithme décrit dans le chapitre précédent. Les données sont d'abord transformées puis l'algorithme de fouille de données est appliqué.
Enfin, à partir des résultats obtenus, une étape de transformation inverse est réalisée de manière à présenter les résultats sous la forme de transactions.
La figure 7.2 illustre la nouvelle base de données obtenue après l'application de la phase de transformation sur notre base de données exemple.

Expérimentation

Pour valider notre approche, nous avons, de la même manière que dans [WL99], réalisé des expériences sur la base de données des films sur Internet (http://us.imdb.com) afin de rechercher des structures typiques de documents concernant le cinéma. Le résultat des expériences est détaillé dans [Lau00]. Cette base de données regroupe, en mai 2000, plus de 200 000 films dont toutes les informations sont organisées sous la forme de pages HTML.
Suite à l'examen de quelques exemples de films, nous nous sommes intéressés à la partie de la base qui concerne les 250 meilleurs films afin d'en extraire les informations concernant les acteurs.

Après avoir récupéré les données concernant les acteurs, nous avons extrait des pages HTML les informations significatives des pages concernant la structure des documents de type acteur. Cette étape a été réalisée en partie automatiquement à l'aide d'un parser mais également manuellement dans la mesure où le contenu des pages ne permettait pas d'extraire, sans ambiguïté, les structures sous-jacentes. Au total nous avons récupéré une base de 500 acteurs dont la profondeur maximale était de 5.

En appliquant les algorithmes de recherche, nous avons par exemple, pour un support de 50%, la structure commune suivante :

$$\{actor : \{name, dateofbirth, filmographyas : \{title, notabletv\}\}\}$$

indiquant que pour au moins 250 acteurs de la base, un acteur possède un nom, une date de naissance et une filmographie. Dans cette filmographie, il apparaît à la fois dans un film mais il a également

effectué des apparitions à la télévision.

7.3 Discussion

Pour évaluer l'approche dans le cadre des transactions imbriquées, nous avons effectué différentes expérimentations sur des jeux de données synthétiques obtenus en modifiant le générateur d'IBM. Les expériences ont montré que l'approche proposée était efficace. Néanmoins, le générateur utilisé ne nous a pas permis de pousser plus en avant certaines expériences. Par exemple, comment se comporte notre algorithme dans le cas de données vraiment très imbriquées (niveau d'imbrication supérieur à 10) ? Les premiers résultats nous ont permis de valider l'approche sur des niveaux d'imbrication de profondeur 6. Bien entendu, avoir des niveaux profonds implique comme conséquence première de rechercher des séquences de taille très longue et donc d'étendre les temps d'exécution de l'algorithme. Il s'agit actuellement d'une des limites des approches de recherche de règles d'association ou de motifs séquentiels qui recherchent généralement des séquences fréquentes de taille limitée. Il faut toutefois préciser cependant que dans la majorité des applications visées ces limites sont tout à fait suffisantes. Lors d'applications plus spécifiques, un pré-traitement des données plus efficace permet alors généralement de résoudre ces problèmes. Par exemple dans le cas du suivi de l'exécution de programme, à partir du moment où nous avons une connaissance sur le domaine traité, il est tout à fait possible de ne pas décrire complètement chaque procédure mais plutôt de les regrouper.

quid de la modélisation des données
Dans l'approche présentée, nous avons considéré que, même si les données étaient imbriquées, elles étaient stockées dans une relation unique composée d'un identifiant et des transactions associées. Cette approche est valide dans de nombreuses applications comme les "logs" où les éléments s'enchaînent les uns après les autres. Par contre, que se passe-t-il si les données sont modélisées au travers de plusieurs relations ? Pour illustrer ce problème de modélisation et ses conséquences, considérons la relation suivante qui contient un ensemble d'items simples et des items définis dans une autre relation.

- $t_1 = \{A,A,B,x\}$
- $t_2 = \{C,w,y\}$

Considérons maintenant que A, B et C sont respectivement définis par $A =< w,x >$, $B =< x,y >$, $C =< x,y,z >$.
Une approche de résolution triviale de ce problème est de remplacer directement A,B et C dans les transactions de manière à obtenir :

- $t_1 = \{< w,x >, < w,x >, < x,y >,x\}$
- $t_2 = \{< x,y,z >,w,y\}$

Cependant, cette approche est loin d'être efficace dans la mesure où elle engendre de nombreux problèmes de mise à jour.
Il est intéressant de souligner que cette modélisation des transactions est utilisée dans l'approche de recherche de régularités proposée dans [WL97a]. Pour résoudre ce problème, les auteurs considèrent que les transactions sont stockées sur le disque alors que les opérations comme A, B ou C sont stockées en mémoire centrale. Cette approche a pour avantage bien entendu de supprimer les problèmes de mise à jour de la base de transaction et d'optimiser les recherches à l'aide des informations stockées en

mémoire. Cependant, la limite principale de cette approche est que si le nombre d'opérations stockées en mémoire est trop grand, il est nécessaire de les retransférer sur disque et de faire une lecture sur le disque pour chaque appel de l'opération.
Concernant l'extraction de connaissances de données semi-structurées issues du Web, les premiers résultats obtenus sur la base de données internationale des films ont montrés que l'approche était adaptée. Cependant, la difficulté essentielle que nous avons rencontrée réside dans le fait qu'il est indispensable de définir un "parser" adapté à chaque type d'application. Dans le cas des films, la difficulté est étendue par la différence de représentation de la structure des films dans la base.

Bien que l'approche proposée ait été validée à l'aide des acteurs, nous avons constaté au cours de nos expériences, qu'il existe des applications (typiquement, celle du cinéma) où il est possible d'améliorer la recherche de régularités. L'amélioration est basée sur le fait qu'il est possible d'optimiser la génération des candidats, i.e. ne générer que des candidats qui sont forcément inclus au moins une fois dans la base et donc de limiter le temps de comparaison entre les transactions et les candidats. Ceci peut être réalisé en parcourant une première fois la base et en stockant chaque séquence dans une structure adaptée. Cette optimisation cependant n'est réalisable que dans le cas d'applications très particulières contenant peu d'items et dont les séquences fréquentes sont nombreuses.

Bien que la problématique de la recherche de régularités dans des bases de données d'objets complexes n'ait pas encore été beaucoup étudiée dans les travaux actuels, il est indéniable qu'elle apparaisse comme le sujet d'une préoccupation forte dans les années à venir. En effet, avec la popularité sans cesse croissante du Web, le nombre d'objets semi-structurés ou d'applications réparties augmente très rapidement. Malheureusement la prise en compte de ces éléments n'est pas encore réalisée de manière satisfaisante : comment interroger une base de données semi-structurée ? comment analyser efficacement les déclenchements d'opérations réparties dans le cas de systèmes de bases de données actives couplées à un data warehouse ?. Les solutions proposées sont généralement basées sur le fait que la structure complexe est déjà connue à l'avance (par exemple les langages de requêtes pour les données semi-structurées). La solution envisagée dans ce chapitre est une réponse partielle à ces problèmes et à ceux des travaux actuels sur les données semi-structurées qui mettent en avant la nécessité d'utiliser des approches de fouille de données pour permettre une meilleure appréhension des données manipulées [Wor97].

Troisième partie

Un domaine particulièrement adapté : l'analyse du comportement des utilisateurs d'un site web

Chapitre 8

Web Usage Mining

En définissant un système d'extraction de connaissances, nous montrons qu'il est possible d'analyser efficacement le comportement des utilisateurs sur un site Web. L'approche présentée, basée sur les algorithmes définis dans le chapitre 4, offre la possibilité d'intégrer des contraintes de temps dans le processus. Cette particularité est garante d'offrir une meilleure vision dans le temps de l'usage d'un serveur Web.

Pour compléter l'approche proposée nous décrivons succinctement une expérience menée pour modifier dynamiquement la structure d'un site Web de manière à adapter le contenu des pages au profil courant des usagers.

8.1 WebTool : un système pour analyser le comportement des utilisateurs

La démarche que nous proposons vise aussi bien l'extraction de règles d'associations que de motifs séquentiels dans le cadre du Web mining. Le système WebTool est décrit dans [MPC99b, MPC99c]. Ces principes généraux, illustrés par la figure 8.1, sont similaires à ceux du processus d'extraction de connaissances exposés dans [FPSSU96]. La démarche se décompose en trois phases principales. Tout d'abord, à partir d'un fichier de données brutes, un pré-traitement est nécessaire pour éliminer les informations inutiles. Dans la deuxième phase, à partir des données transformées, des algorithmes de fouille de données sont utilisés pour extraire les itemsets ou les séquences fréquents. Enfin, l'exploitation par l'utilisateur des résultats obtenus est facilitée par un outil de requête et de visualisation.

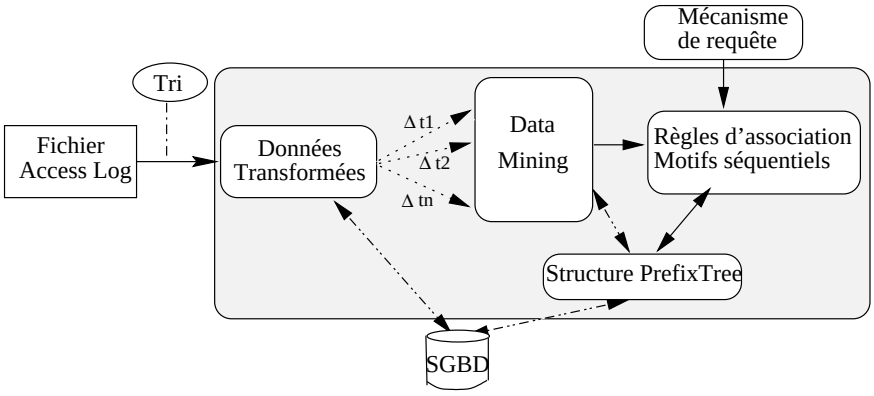

FIG. 8.1 – *Principe généraux de la démarche*

Les différentes phases introduites sont détaillées dans les paragraphes suivants.

8.1.1 Pré-traitement des données

Deux types de traitements sont effectués sur les entrées du serveur log. Tout d'abord, le fichier access log est trié par adresse et par transaction. Ensuite une étape d'élimination des données "non-intéressantes"

est réalisée.

La plupart des outils d'analyse du Web profitent de cette étape pour éliminer des requêtes concernant des pages possédant des graphiques, des vidéos, des scripts CGI ou du son (par exemple, en éliminant les fichiers suffixés par *.GIF, .JPEG*) ou bien les requêtes issues d'agents ou de testeurs de liens. Même si WebTool offre également cette possibilité, nous préférons conserver ces informations comme dans le système WebLogMiner [ZXH98, Coo00]. En effet, ces données apportent des renseignements intéressants concernant aussi bien la structure du site Web, la motivation d'un utilisateur ou les performances du trafic. Par exemple, le fait de conserver des requêtes générées par des agents est utile pour analyser le comportement de l'agent et comparer le trafic généré par ces agents avec le reste du trafic [ZXH98]. De la même manière, si un utilisateur pose des requêtes pour des pages graphiques, certaines de ces requêtes sont importantes pour appréhender les actions de l'utilisateur.

Bien entendu, un tel choix nécessite la mise en œuvre d'algorithmes d'extraction efficaces car la taille du fichier access log reste très grande (au cours de nos expériences, nous avons constaté que la suppression des URL concernant des images réduisait la taille du fichier log de 40% à 85%). En outre, il est nécessaire d'offrir à l'utilisateur un outil de visualisation pour éliminer les données qui ne l'intéressent pas lors d'une analyse particulière.

Au cours de la phase de tri et afin de rendre plus efficace le traitement de l'extraction de données, les URL et les clients sont codés sous forme d'entiers. Toutes les dates sont également traduites en temps relatif par rapport à la plus petite date du fichier.

L'étape de pré-traitement des données offre également la possibilité de regrouper les entrées qui sont suffisamment proches. Contrairement au problème du supermarché où chaque transaction est définie comme un ensemble d'achats effectués par un client, les entrées dans le fichier log sont toutes des transactions séparées. De la même manière que [CMS97], nous proposons de regrouper les entrées qui sont suffisamment proches en utilisant une contrainte de temps (Δt) spécifiée par l'utilisateur. La figure 8.2 illustre un exemple de fichier obtenu après la phase de pré-traitement.

Client	Date	URL traduite
1	01	`10,30,40`
1	02	`20,30`
2	11	`10`
2	12	`30,60`
2	23	`20,50`
3	01	`10,70`
3	12	`30`
3	15	`20,30`

FIG. 8.2 – *Exemple de fichier résultat issu de la phase de pré-traitement*

8.1.2 Outils d'Extraction

A partir des données transformées issues de la première étape, deux techniques d'extraction de connaissance peuvent être appliquées selon les besoins de l'analyste. Etant donné que le problème de la recherche de règles d'association est une réduction du problème de la recherche de motifs séquentiels, le principe retenu par WebTool est d'utiliser la même structure et les mêmes algorithmes pour obtenir des règles d'association. L'adaptation de PSP est prise en compte en considérant que toutes les transactions ont eu lieu en même temps. L'adaptation de PSP est détaillée dans [MPC99a]. L'intégration de la fouille incrémentale est définie dans [MPT99b]. Ainsi, lors de l'application de PSP, les temps des différentes transactions ne sont plus considérés et les résultats obtenus ne sont plus des séquences fréquentes mais des itemsets fréquents. La génération des règles basées sur ces itemsets est réalisée par l'outil de visualisation.

8.1.3 Outils de Visualisation

Devant le très grand nombre de motifs ou d'itemsets obtenus, il apparaît indispensable de proposer un mécanisme de requête pour offrir à l'utilisateur la possibilité de mieux analyser les informations découvertes lors la phase précédente. Un tel mécanisme peut intervenir de deux manières différentes :

1. lors de la phase d'obtention des règles d'association, i.e. après application des algorithmes de recherche de motifs ou d'itemsets fréquents. Il s'agit donc d'extraire des règles à partir des connaissances acquises par les outils d'extraction.
2. lors de l'extraction pour optimiser les temps de traitement réduisant la portion de base à extraire. Ce principe rejoint, par exemple, la phase de pré-traitement où les URL concernant des pages graphiques peuvent être éliminées de la base.

Dans le système WebTool nous privilégions, pour le moment, la première approche dans la mesure où nous considérons que l'utilisateur peut vouloir affiner ces recherches sans recommencer complètement le processus d'extraction.

A l'heure actuelle, de nombreux outils d'extraction proposent d'interroger les résultats via un langage de requête graphique [HFW+96]. Le système WebTool s'inspire de ces travaux et propose à l'utilisateur la possibilité de spécifier le type de règles désiré de la même manière que les "*templates*" utilisés dans le système Tasa [MTV95, MTV97].

8.2 Modification dynamique de la structure hypertexte

Conjointement au système WebTool, nous avons développé un générateur de création de liens dynamiques dans des pages Web à partir des règles obtenues lors du processus d'extraction. Cette composante est détaillée dans [MPT99c]. Le but du générateur est de reconnaître un utilisateur et de vérifier son parcours dans les pages d'un serveur. Lorsque celui-ci correspond à une règle d'association ou à un motif séquentiel déterminé par WebTool, les pointeurs des pages sont dynamiquement mis à jour.

L'architecture fonctionnelle de l'approche est décrite dans la figure 8.3. Le serveur Web (démon *http*)

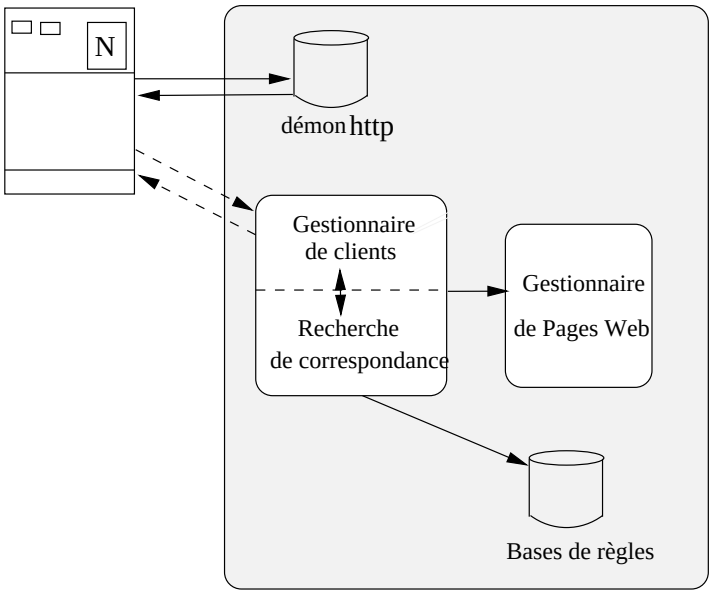

FIG. 8.3 – *Architecture générale*

réagit à l'envoi d'une requête par un client en lui retournant une page contenant une applet. Celle-ci est chargée de se connecter au serveur de clients (*gestionnaire de clients*) pour lui transmettre l'adresse Internet du client et la page demandée. A l'heure actuelle, le gestionnaire de clients est une application java qui fonctionne sur la même machine que le serveur *http* pour permettre à l'applet l'utilisation d'un mécanisme client/serveur.
Lors de la réception de l'adresse et de la page, le gestionnaire de clients examine le comportement du client via le module de recherche de correspondances qui est chargé de vérifier si le comportement du client est en adéquation avec une règle préalablement extraite par le processus de fouille de données.

Lorsqu'une entrée est jugée significative par le gestionnaire de correspondance, la page demandée par l'utilisateur est modifiée par le gestionnaire de pages en ajoutant dynamiquement des liens vers les conséquents de la règle reconnue. L'applet récupère alors l'URL d'accès à cette page et l'affiche sur le navigateur. Si aucune règle ne correspond au comportement du client, l'URL vers la page demandée est retournée à l'applet pour qu'elle puisse l'afficher.

8.3 Discussion

Le système WebTool aborde la problématique de la recherche de comportement d'usager sur un site Web de manière particulièrement originale. Le double objectif ayant guidé cette démarche est d'une part la proposition d'un système capable d'offrir à l'utilisateur final aussi bien une précision sur les pages corrélées que sur le comportement des usagers à travers le temps et d'autre part l'utilisation des résultats obtenus pour modifier dynamiquement un site.

L'approche proposée ne vise pas l'exhaustivité mais se veut suffisamment générique pour prendre en compte de nouveaux types d'algorithmes. Par exemple, une composante segmentation serait particulièrement utile dans un tel système de manière à permettre de regrouper les différents utilisateurs par catégorie. Cependant, en proposant une composante de recherche de motifs séquentiels et de règles d'association nous répondons en partie à la problématique de l'analyse du comportement des utilisateurs.

Outre, le fait que la proposition soit basée sur des algorithmes efficaces, ce qui est particulièrement important dans un tel contexte étant donné les quantités de données manipulées (la plupart des approches de recherche de motifs utilisées sont basées sur des extensions d'algorithmes de règles d'association [Coo00] qui ont pourtant montré leur limite pour les motifs séquentiels [AS95, SA96b]), elle offre également une prise en compte des contraintes de temps. Cette composante, inexistante dans les autres propositions, permet d'offrir à l'utilisateur final une analyse plus précise du comportement des clients.

En intégrant la gestion de recherche incrémentale, nous étendons le caractère original de la proposition. En effet, l'une des particularités d'un serveur Web est d'avoir des données qui évoluent constamment. La seule approche, à l'heure actuelle qui permet de prendre en compte ces évolutions est l'approche du projet WebLogMiner [ZXH98], mais au travers d'un système OLAP. L'intégration de la composante recherche incrémentale se fait de manière tout à fait naturelle dans le système WebTool proposé et permet de maintenir une base de connaissance extraite.

Les expériences menées sur la modification dynamique de la structure hypertexte d'un serveur conforte le fait que la proposition permet de répondre à une problématique très actuelle des gestionnaires de serveur Web : comment maintenir les clients sur le serveur ? L'une des solutions préconisée est justement d'essayer d'adapter le contenu du serveur pour qu'il soit le plus en adéquation avec le comportement des utilisateurs. Notre proposition est un premier élément de réponse à ce problème.

L'approche proposée ouvre également la voie à un autre type d'application : la détection d'intrusion sur des systèmes [LSM99]. En effet, dans ces applications, les données manipulées correspondent à des fichiers ordonnés dans le temps (tcpdump, BSM), et elles abordent deux problématiques principales : reconnaître les comportements usuels de ceux correspondant à une fraude et être capable de reconnaître en temps réel une tentative de fraude dans le système. Pour résoudre la première problématique les auteurs préconisent l'utilisation d'algorithme de recherche de motifs séquentiels efficaces dans la mesure où la prise en compte du temps est un élément particulièrement crucial dans l'examen du comportement des utilisateurs du système.

Chapitre 9

Web Usage Mining Inter-sites

Cette section a pour objectif la manipulation et l'intégration de trois problématiques, afin d'en extraire un concept cohérent et utile pour le Web Usage Mining Inter-Sites. La première de ces problématiques est la recherche de **motifs séquentiels**, en raison de son adaptation pour l'analyse de fichiers logs. La seconde problématique est celle du **Web Usage Mining**, dans la mesure où le Web Usage Mining Inter-Sites est lui même un problème d'analyse du comportement des utilisateurs d'un site web. L'originalité de notre approche vient du fait que la troisème problématique mise en jeu est la recherche **incrémentale** de motifs séquentiels. Nous avons en effet constaté une forte analogie entre l'aspect incrémental de la recherche de motifs séquentiels, d'un côté, et la mise en commun des fichiers access log de deux sites partenaires, d'un autre côté. Cette analogie, afin de devenir une véritable translation, mérite quelques réflexions sur le support minimum. La sous-section 9.1 présente ces réflexions, alors que la section 9.2 décrit l'algorithme qui en résulte.

9.1 Réduction

La première méthode que l'on peut envisager de mettre en place pour résoudre le problème du Web Usage Mining Inter-Sites, consiste à construire le fichier $Log+$ (fichier log du site virtuel $A \cap B$) selon l'algorithme 9.1. Une fois ce fichier obtenu, l'étape de fouille de données va alors analyser les données qu'il contient afin d'extraire les comportements fréquents des utilisateurs que le site A a redirigé vers le site B. Ce paragraphe propose une approche plus performante, issue d'une analyse théorique mettant en jeu les propriétés des fréquents d'une part, et le support minimum d'un processus de fouille de données permettant de les obtenir sur le fichier log du site A, d'autre part.

fonction *construireLog+*
Input : Log_A et Log_B les fichiers log des sites A et B.
Output : $Log+$ le fichier log du site virtuel $A \cap B$.
for $C \in Log_A$ **do**
  if *dernierObjet(C)="To_ B"* **then**
    InscrireSequence(C,$Log+$); //Inscrire la séquence de C dans $Log+$
  endif
endfor
//Le fichier obtenu à la fin de cette première boucle sera désigné par $Log+_A$
for $C \in Log_B$ **do**
  if *premierObjet(C)="From_ A"* **then**
    InscrireSequence(C,$Log+$); //Inscrire la séquence de C dans $Log+$
  endif
endfor
//La partie ajoutée au fichier $Log+$ à la fin de cette seconde boucle sera désignée par $Log+_B$

Algorithme 9.1: *Création du fichier Log+*

Adaptation de la problématique de la fouille de données incrémentale

Les sites concernés par les applications du Web Usage Mining Inter-Sites analysent leurs fichiers access log, pour en extraire des comportements révélateurs, avec différents supports et de façon régulière, comme tout site le ferait dans le cadre du Web Usage Mining Intra-Site. Ce paragraphe présente une méthode destinée à exploiter les résultats obtenus de manière locale sur les sites mis en jeu, afin d'optimiser le processus de Web Usage Mining Inter-Sites.

La définition 13 présente la notion de $Log+_W$. Pour un site W mis en jeu dans un processus de Web Usage Mining Inter-Sites, $Log+_W$ représente la partie du fichier $Log+$ qui correspond aux navigations des clients du site W sur le site W. L'exemple 50 donne une illustration de ce concept.

Définition 13 Soit $Log+$ le fichier log du site virtuel $A \cap B$ obtenu à partir des fichiers log des sites A et B. $Log+_A$ représente la partie du fichier $Log+$ concernant la navigation des clients sur le site A et $Log+_B$ la partie du fichier $Log+$ concernant la navigation des clients sur le site B.

Exemple 50 *Si l'on considère à nouveau le fichier Log+ obtenu à la figure 2.14, alors le fichier* $Log+_{SF1}$ *qui lui correspond est illustré par la figure 9.1.*

Client	d1	d2	d3	d4	d5
$C1$	a	b	c	d	To_SV
$C2$	a	c	d	e	To_SV

FIG. 9.1 – *Fichier* $Log+_{SF1}$

Pour la suite de ce paragraphe, L^x_{DB} désigne l'ensemble des séquences fréquentes maximales sur DB avec un support minimum x (i.e. $\forall s \in L^x_{DB}$, soit n le nombre d'apparitions de s sur DB, $n > x.|DB|$). De même, si une séquence s a un nombre d'apparitions n sur $|DB|$ alors son support est de $\frac{n}{|DB|}$.

propriété 5 $\forall x \in]0..1]$, soit $s \in L^x_{Log+_A}$, s se termine par To_B.

La propriété 5 se vérifie de façon directe car 100% des séquences de $Log+_A$ se terminent par To_B. A partir de cette observation, on peut déduire que tout fréquent extrait sur le fichier $Log+_A$ se termine obligatoirement par l'objet To_B.
Le théorème 7 repose sur cette propriété et sur une manipulation des différents supports considérés lors de la phase d'extraction sur le fichier log de A et dans le cadre du Web Usage Mining Inter-Sites.

Théorème 7 Soient x_1 et x_2, deux supports tels que $x_2 = \frac{x_1.|Log+_A|}{|Log_A|}$ et soit $FreqLog+_A = \{s \in L^{x_2}_{Log_A}\}$ avec s se terminant par To_B, alors $FreqLog+_A = L^{x_1}_{Log+_A}$.

preuve : Prouvons que si s est une séquence fréquente sur $Log+_A$ avec un support x_1 (i.e. $s \in L^{x_1}_{Log+_A}$), alors $s \in FreqLog+_A$ avec un support $x_2 = \frac{x_1.|Log+_A|}{|Log_A|}$.

Le nombre d'apparitions de s sur $Log+_A$ est supérieur à $x_1.|Log+_A|$. Ce qui est également le nombre d'apparition minimum de s sur Log_A car $Log+_A \subset Log_A$. s a donc sur Log_A un support $x2 > \frac{x_1.|Log+_A|}{|Log_A|}$. De plus, d'après la propriété 5, s se termine par To_B. Donc $s \in FreqLog+_A$ car elle satisfait les règles de construction de $FreqLog+_A$. Donc $L^{x_1}_{Log+_A} \subseteq FreqLog+_A$.

Prouvons enfin que si $s \in FreqLog+_A$ avec un support $x_2 = \frac{x_1.|Log+_A|}{|Log_A|}$, alors s est fréquente sur $Log+_A$ avec un support x_1 (i.e. $s \in L^{x_1}_{Log+_A}$) .

Si $s \in FreqLog+_A$ alors s se termine par To_B (par construction de $FreqLog+_A$) et le nombre de séquences qui supportent s dans Log_A est le même dans $Log+_A$ (par construction de $Log+$). Soit n le nombre d'apparitions de s sur $Log+_A$, $n > x_2.|Log_A|$ donc $n > \frac{x_1.|Log+_A|}{|Log_A|}.|Log_A|$ et donc $n > x_1.|Log+_A|$. Donc s est fréquente sur $Log+_A$ avec un support de $x1$ (i.e. $s \in L^{x_1}_{Log+_A}$). Donc $FreqLog+_A \subseteq L^{x_1}_{Log+_A}$.

Comme $L^{x_1}_{Log+_A} \subseteq FreqLog+_A$ et $FreqLog+_A \subseteq L^{x_1}_{Log+_A}$, alors $FreqLog+_A = L^{x_1}_{Log+_A}$ □

Le théorème 7 exprime le fait que si l'on envisage de lancer un processus d'extraction sur le fichier $Log+$ avec un support x_1, alors il existe un support x_2 pour lequel le fichier Log_A recèle tous les fréquents du fichier $Log+_A$ calculés avec le support x_1. Pour retrouver ces fréquents, il suffit alors de prendre parmi les fréquents de Log_A avec le support x_2, ceux qui se terminent par To_B. Une fois ces fréquents obtenus (par une simple sélection parmi les fréquents déjà calculés par le site A) le processus de Web Usage Mining Inter-Sites dispose alors d'une information non négligeable, puisqu'il peut l'utiliser en tant que résultat d'un premier traitement sur le fichier $Log+_A$ et considérer $Log+_B$ comme un ajout à $Log+_A$ qui aboutit sur $Log+$. Ce raisonnement nous conduit alors vers la conclusion suivante : dans la mesure où l'on dispose déjà (grâce aux calculs fait par le site A, à une adaptation du support et à une sélection parmi les résultats) des résultats correspondants à un processus de fouille de données sur $Log+_A$ avec le support x_1, nous pouvons les utiliser comme données en entrée d'un processus de fouille de données incrémentale pour calculer l'ensemble des fréquents de $Log+$ avec le support x_1.

Ce raisonnement est alors motivé (et sa mise en œuvre encouragée) par les temps de calcul améliorés que nous garantissent des méthodes incrémentales d'extraction de motifs séquentiels [PZOD99, MPT99b].

9.2 WUMIS :Un algorithme pour le Web Usage Mining Inter-Sites

Dans ce paragraphe, après un rappel de la problématique de l'extraction incrémentale de motifs séquentiels, nous proposons une façon efficace de la résoudre via l'algorithme ISE+, une extension de l'algorithme ISE [MPT99b]. Nous présentons ensuite l'algorithme WUMIS qui est basé sur l'algorithme ISE+ et sur les résultats du théorème 7 pour résoudre le problème du Web Usage Mining Inter-Sites.

L'algorithme WUMIS

Pour résoudre la problématique du Web Usage Mining Inter-Sites, nous proposons d'exploiter le théorème 7, gràce à l'algorithme WUMIS. Cet algorithme décrit dans la figure 9.2 procède en trois phases. Tout d'abord construire le fichier $Log+$. Ensuite calculer le support x_2 et construire l'ensemble des fréquents calculés sur Log_A par le site A avec ce même support et se terminant par To_B. Enfin, cet ensemble étant prouvé identique à celui des séquences fréquentes sur $Log+_A$, WUMIS l'utilise comme paramètre pour l'algorithme ISE+, qu'il va alors faire fonctionner sur le fichier $Log+$, avec pour incrément $Log+_B$.

Algorithm WUMIS
Input: Log_A et Log_B les fichiers log des sites A et B. x_1 le support minimum.
Output: $L^{x_1}_{Log+}$ l'ensemble des séquences fréquentes sur $Log+$
Method:

//Première phase : $Log+$
$(Log+, Log+_A, Log+_B) \leftarrow$ construireLog+(Log_A, Log_B);
//Seconde phase sur Log_A
$x_2 \leftarrow \frac{x_1.|Log+_A|}{|Log_A|}$;
$L^{x_1}_{Log+_A} \leftarrow s/(s \in L^{x_2}_{Log_A}$ et s se termine par $To\_B)$;
//Dernière phase sur $Log+$
$L^{x_1}_{Log+} \leftarrow$ ISE+ $(Log+_A, L^{x_1}_{Log+_A}, Log+_B, x_1)$;

end Algorithm WUMIS

FIG. 9.2 – *l'Algorithme* WUMIS

9.3 Discussion

Nous avons proposé, dans ce chapitre, une réduction de la problématique incrémentale des motifs séquentiels vers celle du Web Usage Mining Inter-Sites, qui exploite un calcul sur le support minimum. Les performances de notre algorithme sont alors dépendantes de celle d'ISE, qui est conçu pour cette recherche et dont les expérimentations sont décrites à la section 6.4. Cette dépendance adopte un caractère majeur, dans le sens où, sorti de l'application d'ISE, l'algorithme WUMIS s'exécute en un temps minimum sur la base de données (calcul du support, sélection des fréquents qui le respectent, indexation des clients concernés par le partenariat).

Nous sommes cependant conduits à nous demander si la prise en compte de plusieurs (plus de deux) sites partenaires pourra garder la même efficacité. Cela pose la question des ajouts successifs dans le contexte de l'incrémental, et des performances associées. Ainsi, nous pourrions proposer une méthode pour le Web Usage Mining Inter-Sites qui prenne en compte trois, puis jusqu'à n sites partenaires.

Chapitre 10

Web Usage Mining temps réel

10.1 Puissance de calcul

Pour contourner le problème lié à la puissance de calcul disponible, nous proposons de faire évaluer les candidats par chaque "acteur" de la base de données : les machines des utilisateurs connectés. En effet chaque utilisateur est actuellement en train d'utiliser un navigateur et donc une puissance de calcul qui lui est locale afin de naviguer sur ce site. Cette puissance de calcul est disponible si l'on offre l'architecture nécessaire à l'exploiter. Ainsi la base de données représentée à l'étape 4 de la figure 2.20 de la section 2.8 peut être remplacée par l'ensemble des machines connectées qui la composent.

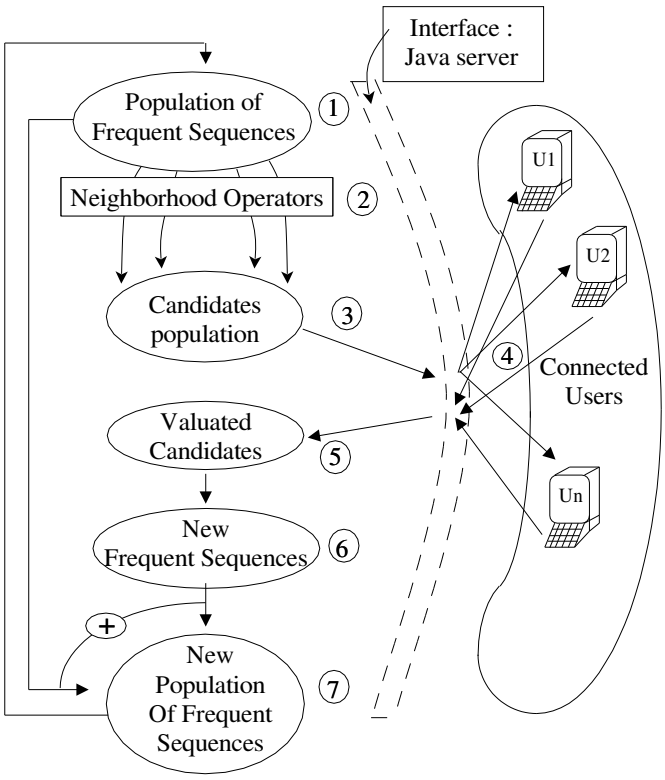

FIG. 10.1 – *Proposition d'une architecture client/serveur dans laquelle le client effectue des calculs pour le serveur.*

Cette opération se fait en fournissant à la méta-heuristique une interface, qui simule une passe sur la base de données en contactant chaque machine connectée (Fig. 10.1). Chaque machine ainsi contactée va comparer sa propre séquence de navigation avec l'ensemble des candidats qu'elle vient de recevoir et renvoyer les résultats de cette comparaison. Cela revient à comparer chaque séquence de la base (qui regroupe les séquences des machines connectées) avec les séquences candidates, mais en un temps qui satisfait les conditions demandées par une fonction objective. Dans le cas de l'étape 4 de la figure 2.20,

la complexité de ce calcul est sensiblement $O(n \times m \times k \times l)$, avec :

- n le nombre de séquences contenues dans le fichier access log (taille de la base),
- m le nombre de candidats dans la solution proposée,
- k la taille moyenne des séquences de la base,
- l la taille moyenne des candidats.

Nous utilisons en effet un algorithme issu de la programmation dynamique pour calculer la plus longue séquence commune aux deux séquences à comparer ($O(longueurSquence1 \times longueurSquence2)$ [CLR94]) pour chaque séquence candidate et chaque séquence de la base.

En revanche, dans le cas de la figure 10.1, si cette complexité reste la même les calculs se trouvent répartis. Chaque calcul voit alors sa complexité exprimé comme : $O(m \times k \times l)$ avec k la taille de la séquence de navigation sur la machine contactée et l la taille moyenne des candidats. Le nombre d'appel à ce calcul reste n, mais quelque soit le nombre de machines à contacter, le temps d'exécution necessaire à évaluer une population est $\mathbf{max}(T1 + T2 + T3)$, avec :

- $T1$: le temps nécessaire pour envoyer les candidats à une machine n
- $T2$: le temps de calcul pour comparer la séquence de n avec les candidats envoyés
- $T3$: le temps nécessaire pour que n renvoie ses résultats au serveur

Une différence fondamentale avec la fouille de données classique provient du fait de ne travailler que sur les machines connectées. En effet, toutes les machines figurant dans le fichier access log ne sont pas forcément connectées. Cela fait partie des avantages de cette méthode, qui fournit des résultats parfaitement adaptés aux utilisateurs connectés car ces résultats sont très fortement orientés par le comportement des utilisateurs de leur catégorie : les connectés (ou qui étaient connectés très récemment). En effet, un fichier access log peut s'étaler sur plusieurs jours, semaines ou mois. Les comportements issus de l'analyse d'un tel fichier sont des comportements à long terme, mais une telle analyse ne peut fournir de comportements à court terme correspondants aux comportements les plus récents (voire actuels).

10.2 HDM : une architecture client/serveur/moteur

Pour mettre en oeuvre la méthode présentée sur la page précédente, nous avons mis en place l'architecture décrite à la figure 10.2. Quand une machine demande l'accès à une page sur le serveur HTTP, celui ci lui renvoie la page demandée avec une applet Java. Cette applet sera alors chargée de contacter le serveur Java, afin de recevoir les calculs à effectuer. Au début du cycle de vie du site, le premier client qui vient se connecter est le moteur de calcul, sur lequel se trouve l'algorithme stochastique. Le moteur va donc régulièrement envoyer au serveur des calculs à distribuer sur les machines connectées.

Le premier problème à résoudre consiste à maintenir les supports pour les différentes pages accédées en temps réel. L'exemple 51 donne une illustration de la méthode utilisée dans ce but.

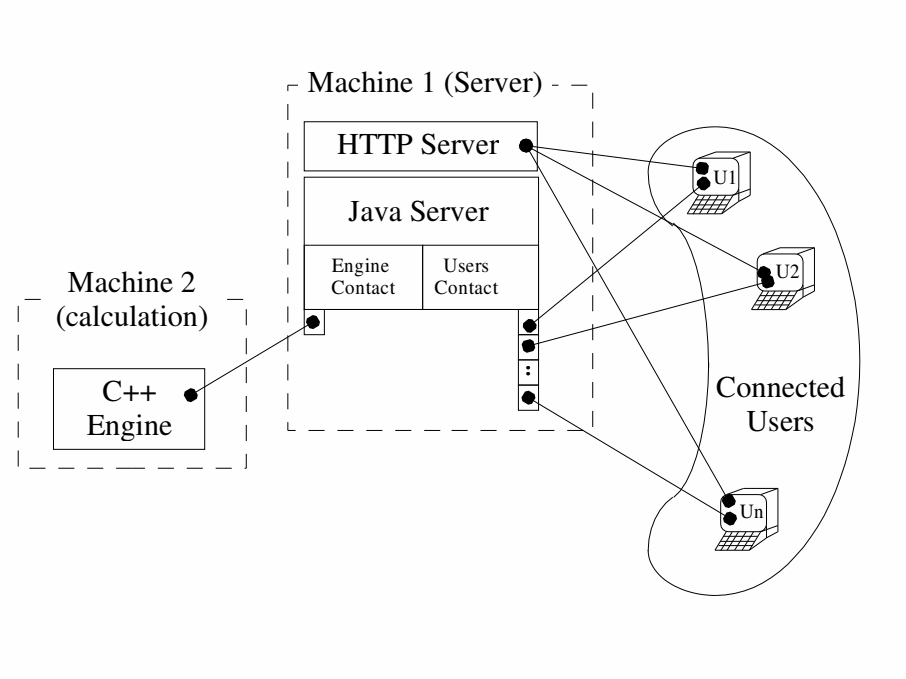

FIG. 10.2 – *Connexion entre le générateur de candidats et les clients via le serveur Web*

Exemple 51 *Considérons que 2 utilisateurs (U_1 et U_2) sont actuellement connectés sur le site. Leurs séquences de navigation sont* `< (1) (2) (1) (3) (5) >` *pour U_1 et* `< (1) (2) (3) (4) (5) >` *pour U_2. Les supports des pages à ce moment précis sont inscrits dans la table suivante :*

Page	*1*	*2*	*3*	*4*	*5*
Support	*100%*	*100%*	*100%*	*50%*	*100%*

Cette table est maintenue par le serveur Java et retransmise régulièrement au moteur C++ qui va utiliser les fréquences renseignées afin de générer ses candidats. Considérons un troisième utilisateur (U_3) se connectant au site Web et demandant les pages 1, 3 et 2. Chaque page demandée (si elle n'apparaît pas déjà dans la séquence de navigation de l'utilisateur) sera signalée au serveur Java, dans le but de maintenir le support de chaque page, selon les connexions des utilisateurs. Considérons à présent que U_3 demande la page 3 de nouveau. Cette page fait déjà partie de la séquence de navigation de cet utilisateur et cette action ne sera donc pas signalée au serveur Java. Enfin, U_3 demande la page 7, ce qui est une nouvelle page dans la séquence de navigation de U_3 et a pour conséquence d'informer le serveur Java de cette requête. La séquence de U_3 est désormais `< (1) (3) (2) (3) (7) >` *et les nouveaux supports des pages sont:*

Page	*1*	*2*	*3*	*4*	*5*	*7*
Support	*100%*	*100%*	*100%*	*33%*	*66%*	*33%*

Avec un support minimum de 80%, le motif de comportement fréquent à trouver est `< (1) (2) (3) >`*. Nous considérons que 3 n'apparaît qu'une seule fois (et non pas deux) dans la séquence de U_3, en*

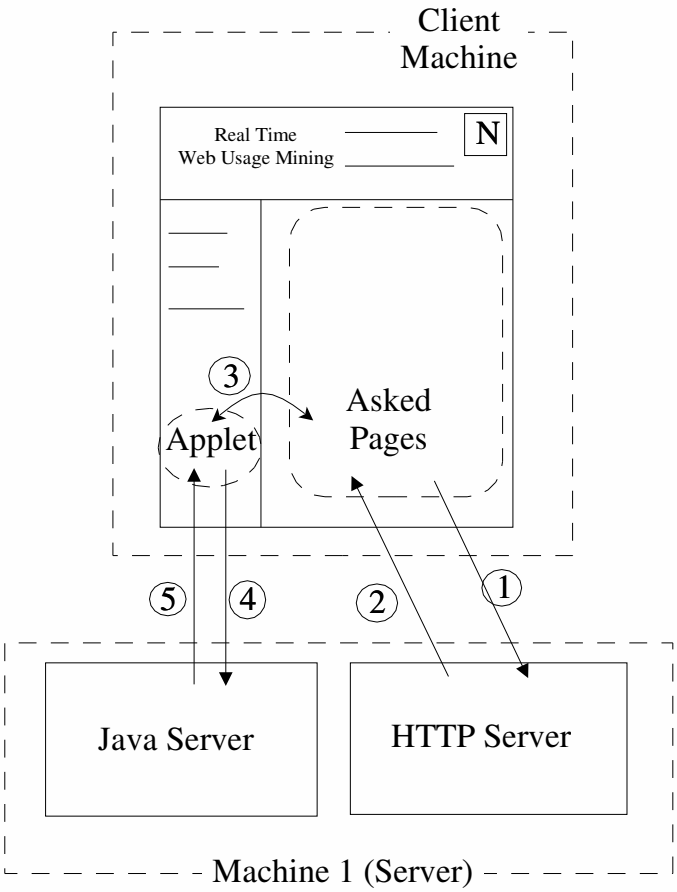

FIG. 10.3 – *Mise à jour en temps réel des supports pour les pages accédées*

raison de la définition d'un item fréquent. En fait, considérer le nombre d'apparition d'un item dans la séquence de navigation d'un utilisateur aurait pour conséquence de biaiser les résultats dans la mesure où un client ayant demandé minSupport fois la page n serait suffisant pour rendre la page n fréquente, même si il est le seul à l'avoir demandée.

L'architecture de la figure 10.3 ne décrit pas seulement la façon dont les pages fréquentes sont découvertes en temps réel, mais donne également une illustration des opérations effectuées au cours de la navigation d'un utilisateur sur le site :

1. Le navigateur demande une page (celle que l'utilisateur veut consulter) au serveur HTTP.
2. Le serveur envoie la page demandée qui à son arrivée contacte l'applet pour être ajoutée à la séquence de navigation de cet utilisateur.

3. L'applet, une fois contactée informe le serveur Java de cette nouvelle page.
4. L'applet renvoie également les résultats des calculs demandés par le serveur.
5. Le serveur contacte régulièrement l'applet pour lui confier des calculs.

Apportons quelques détails sur ce dernier point. Comme le moteur connait les supports des différentes pages, il va utiliser cette connaissance pour générer des candidats. Ces candidats, une fois évalués par la machine de l'utilisateur connecté, vont devenir fréquents ou pas, et les motifs fréquents découverts seront utilisés en combinaison avec les pages fréquentes afin de générer de nouveaux candidats, et ainsi de suite...

La façon dont les nouveaux candidats sont proposés est développée dans la section 10.3. Le moteur C++ est alors basé sur l'algorithme 10.1. Le critère de sélection prend en compte la taille du candidat (c), le support de ce candidat et une évaluation des possibilités d'accepter c même si la séquence ne le "mérite pas". Cette technique permet à HDM de suivre le comportement des utilisateurs car les candidats ne sont pas évalués par une fonction booléenne (i.e. "inclusion ou pas") mais par une fonction qui mesure la taille de la séquence incluse dans le candidat et la compare à la taille totale du candidat comme décrit dans la section 10.3.

fonction *Moteur_Hdm*
Input : JS le serveur Java recquis pour l'architecture
Output : SP les motifs séquentiels correspondants aux comportements fréquents.
while *le site Web est en ligne* **do**
  pagesSupports=getPagesSupports(JS); // Maintenir le support des pages
  candidates=getValuation(JS); // Obtenir les résultats des calculs distribués
  for $c \in$ *candidates* **do**
    if *(support(c) > minSupport OU critere)* **then**
      inserer(c, SP);
    endif
  endfor
  candidates=neighborhoodOperators(*candidates*, *pagesSupport*);
  demandeDistribution(JS, *candidates*);
endwhile
fin fonction *Moteur_Hdm*

Algorithme 10.1: *Le moteur de* HDM

D'un point de vue réseau, les communications demandées par cette méthode restent faibles dans la mesure ou les séquences sont envoyées par des techniques de broadcast et avec une compression à la volée de ratio $\frac{1}{5}^{eme}$ qui est appliquée sur les séquences envoyées (une population de 300 séquences coûte environ 8 Ko sans compression et 1,5 Ko compressée).

10.3 Opérateurs de voisinage et évaluation des candidats

La principale idée, sur laquelle HDM se base, consiste à générer des population de candidats grâce aux motifs fréquents, aux pages fréquentes et aux opérateurs de voisinage, puis d'envoyer ces candidats aux machines connectées afin d'évaluer leur pertinence (ou tout au moins leur distance d'une séquence fréquente). Ces deux phases (opérateurs de voisinage et évaluation des candidats) sont expliquées dans cette section.

10.3.1 Opérateurs de voisinage

Les opérateurs de voisinages présentés dans cette section, ont été validés grâce à une batterie d'expérimentations réalisées sur des bases de données de manière locale (par opposition à l'aspect distribué de la méthode présentée) car seule leur efficacité en terme d'approche des fréquents réels étaient en jeu (voir section 10.4). Nous avons choisi des opérateurs de type génétique, aussi bien que des opérateurs basés sur les propriétés des motifs séquentiels fréquents. Quand nous faisons référence aux "séquences choisies aléatoirement" (ou bien "séquences aléatoires"), nous utilisons une roulette biaisée, telle que les séquences ayant le support le plus haut sont choisies plus fréquemment que les séquences de support plus faible.

Enfin, nous nous sommes efforcés d'évaluer le taux de réussite de nos opérateurs, en calculant la moyenne des séquences retenues par rapport aux séquences proposées, en fin de processus. Ainsi, un opérateur affichant un taux de réussite de 20% est un opérateur qui, en fin de processus, a vu 20% des candidats qu'il propose s'avérer fréquents. Le taux de réussite brut, est un taux de réussite duquel on a décompté les séquences proposée par d'autres opérateurs et déterminées fréquentes (i.e. les séquences fréquentes trouvées grâce à cet opérateur et lui seul).

Nouveaux items fréquents

Quand un nouvel item fréquent apparaît (après avoir été demandé par un ou plusieurs utilisateurs) il est utilisé pour générer toutes les séquences de taille 2 possibles avec les autres items fréquents. Les candidats ainsi générés sont ajoutés à l'ensemble des candidats à envoyer. En raison du grand nombre de candidats générés, cet opérateur n'a qu'un taux de succés de 15%. Cet opérateur reste cependant essentiel car il permet d'obtenir toutes les séquences fréquentes de taille 2, qui sont la base pour les opérateurs suivants.

Ajout d'items

Cet opérateur a pour but de choisir un item aléatoirement parmi les items fréquents, puis d'ajouter cet item à une séquence s choisie aléatoirement, après chaque item de s. Chaque ajout donnant lieu à un nouveau candidat. Cet opérateur génère $longueur(s) + 1$ candidats. Par exemple, avec la séquence $<$ (a) (b) (d) $>$ et l'item c, nous allons générer les séquences candidates $<$ (c) (a) (b) (d) $>$, $<$ (a) (c) (b) (d) $>$, $<$ (a) (b) (c) (d) $>$ et finalement $<$ (a) (b) (d) (c) $>$. Cet opérateur connait un taux de réussite de 20%, mais les séquences trouvées sont très utiles au reste des opérateurs.

Croisement de base

Cet opérateur (largement inspiré des opérateurs génétiques) utilise deux séquences aléatoires afin de proposer deux candidats issus de leur mélange. Par exemple, avec les séquences < (a) (b) (c) > et < (d) (e) (f) >, nous proposons les candidats < (a) (b) (e) (f) > et < (d) (e) (b) (c) >. Cet opérateur affiche un taux de réussite performant (50%) grâce aux séquences fréquentes contenues dans les séquences obtenues par les opérateurs précédents.

Croisement amélioré

Encouragés par les résultats du premier croisement, nous avons développé un nouvel opérateur, conçu pour être une amélioration du croisement de base et reposant sur les propriétés des séquences fréquentes. Cet opérateur a pour but de choisir deux séquences aléatoires, et le croisement ne s'effectue pas au milieu des séquences, mais à la fin du plus long préfixe commun aux deux séquences traitées. Considérons deux séquences < (a) (b) (f) > et < (a) (c) (d) (e) >. Le plus long préfixe commun à ces deux séquences est < (a) >. Le croisement va donc commencer après l'item qui suit a, pour chaque séquence. Dans notre exemple, les deux séquences résultant de ce croisement seront < (a) (b) (c) (d) (e) > et < (a) (c) (b) (f) >. Cet opérateur connait un taux de succès de 35%.

Précisons que les séquences déjà obtenues grâce au croisement de base sont décomptées lors du calcul du taux de réussite de cet opérateur. Ce taux de 35% est donc un apport brut pour les opérateurs précédent (i.e. des séquences que cet opérateur est le seul à fournir).

Dernier croisement

Un dernier opérateur de croisement consiste à améliorer les deux précédent. Il est basé sur le même principe que l'opérateur de croisement amélioré, à cette différence que la seconde séquence n'est pas choisie aléatoirement. En effet la seconde séquence est choisie comme étant celle qui présente le plus long préfixe commun avec la première séquence choisie. Cet opérateur présente un taux de réussite (brut) de 30%.

Extension de séquences

Cet opérateur est basé sur l'idée suivante : les séquences fréquentes sont étendue avec les nouvelles pages demandées et devenues fréquentes. Le principe de base est donc d'ajouter les nouvelles pages à la fin de quelques séquences aléatoires. Cet opérateur affiche un taux de succès de 60%.

La figure 10.4 donne une illustration de quelques opérateurs décrits dans cette section.

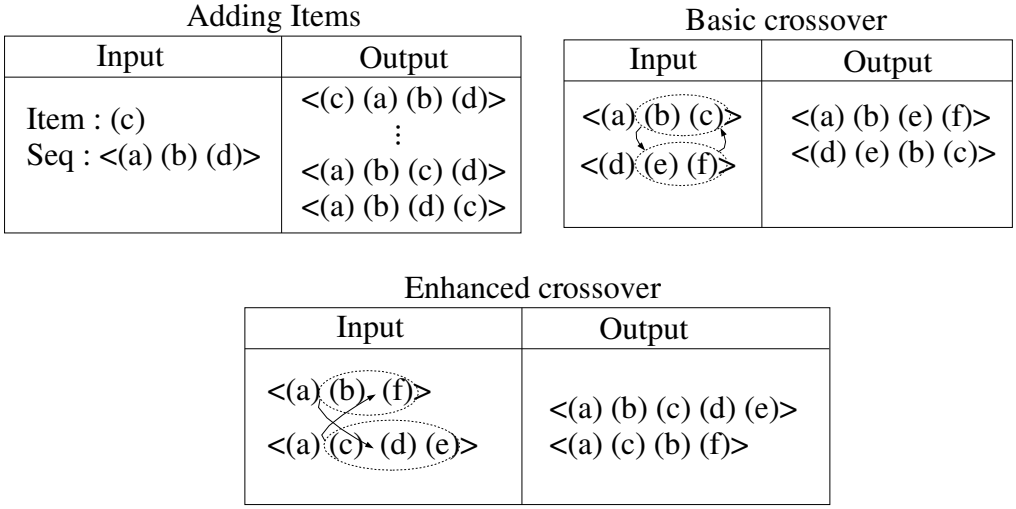

FIG. 10.4 – *Quelques opérateurs conçus pour la recherche de séquences de navigation fréquentes*

10.3.2 Calculs demandés au navigateur client

Quand la génération de candidats est achevée, le moteur C++ envoie l'ensemble des candidats au serveur Java et ce dernier, au travers d'une technique de broadcast, le redistribue aux navigateurs connectés. Le calcul demandé aux navigateurs connectés est alors mis en œuvre par l'applet Java, lue au moment ou l'utilisateur se connecte sur le site, et consiste à comparer chaque séquence de l'ensemble des candidats reçus avec sa propre séquence de navigation. La comparaison consiste à obtenir un pourcentage, qui représente la distance entre le candidat et la séquence de navigation de l'utilisateur. Si la séquence candidate est incluse dans la séquence de navigation, le pourcentage sera de 100% et ce taux va décroitre avec l'apparition de parasites (différences entre le candidat et la séquence de navigation). De plus, dans le but d'obtenir des séquences les plus longues possible, nous utilisons un algorithme qui favorise les séquences les plus grandes, si elles sont incluses dans la séquence de navigation. D'un autre côté, dans le but d'éviter de favoriser les séquences les plus grandes si elles présentent une distance par rapport à la séquence de navigation, cet algorithme a pour but de sanctionner les séquences trop longues si elles ne sont pas incluses (plus la séquences est longue, plus sa distance à la séquence de navigation sera pénalisante).

Pour prendre en compte tous ces paramètres, le calcul effectué par la machine cliente est décrit par l'algorithme 10.2.

fonction *Client*
Input : CS les candidats à évaluer et N la séquence de navigation du client.
Output : P l'ensemble des pourcentages affectés à chaque séquence.
for $S \in CS$ **do**
if *($S \subseteq N$)* **then**
$P[S]$=100+taille(S);// S est incluse dans N, S est favorisée.
endif
if *(taille(S)≤ 2)* **then**
$P[S]$=0; // S de longueur n'est pas incluse, la garder ne présente aucun intérêt.
endif
// Sinon, donner une note à S et défavoriser les distances accrues.
$P[S]=\frac{taille(LCS(S,N))*100}{taille(S)} - taille(S)$
endfor
fin fonction *Client*

Algorithme 10.2: *Algorithme client*

|D|	Taille de la base de données
|C|	Nombre moyen de transactions par client
|S|	Taille moyenne des transactions fréquentes
N	Nombre d'items

TAB. 10.1 – *Parametres*

10.4 Experimentations

Nous proposons tout d'abord de présenter les expérimentations que nous avons menées, afin d'évaluer la qualité des résultats fournis par la méta-heuristique que nous utilisons et sa capacité à fournir ces résultats en temps réel. Dans le but de valider l'approche HDM, nous avons du tester la méthode en local, à l'aide d'une base de données où chaque séquence a été associée à une tâche (visant à simuler le comportement de la machine appartenant à l'utilisateur qui a suivi cette séquence de navigation). Nos expérimentations ont été réalisées sur bases de données dont les caractéristiques sont inscrites dans les tables 10.1 et 10.2.

Nom de la base	D	C	S	N
Access 1	9K	33	15	60
Access 2	11K	42	38	32
Access 3	8K	38	14	60
Access 4	12K	32	11	60

TAB. 10.2 – *Jeu de données synthétiques*

10.4.1 Mesure de la qualité

Afin de mesurer la qualité des résultats fournis par notre algorithme HDM, nous avons évalué, pour chaque population testée, la taille de la plus longue séquence commune (PLSC) entre les candidats de la population et les séquences du résultat réel (obtenu au préalable par un appel à l'algorithme de fouille de données PSP, développé au Lirmm par notre équipe). L'algorithme appliqué à chaque population proposée est donc le suivant (algorithme 10.3) :

fonction *calculQualité*
Input : $candidatePopulation$, une population candidate à évaluer. $PSP\_realResults$ les résultats obtenus par PSP (pour comparer).
Output : $quality$, le pourcentage de qualité obtenu par comparaison avec les résultats de PSP.
sum=0;
for $s1 \in candidatePopulation$ **do**
  $localQuality$=0;
  for $s2 \in PSP\_realResults$ **do**
    $localQuality$=max($localQuality$, (PLSC($s1$,$s2$)/size($s2$))*100);
  endfor
  $sum = sum + localQuality$;
endfor
$quality$=sum/size($candidatePopulation$);
return($quality$)
fin fonction *calculQualité*

Algorithme 10.3: *Algorithme de calcul de qualité*

10.4.2 Valider la qualité des résultats

Expérimentations sur un comportement de type stable:

Le but de ce test est de valider l'algorithme HDM, en fonction de la qualité des résultats fournis pour un type de comportement précis. Pour cela nous avons utilisé une base de données que nous avons confiée à HDM, qui pour chaque population testée sur la base a fait la comparaison entre la population proposée et les résultats réels. Pour des résultats de taille courte (longueur de fréquents : 4), la qualité 100% est atteinte avec 3 populations proposées (c'est le meilleur résultat possible, dans la mesure ou la méthode Générating-Pruning propose (k-1) populations candidates, avec k la longueur du plus grand fréquent). Le tableau suivant résume les résultats obtenus pour des fréquents de longueur 4, sur la base "Access 4" :

Generation	1	2	3
Qualité	50%	75%	100%

Ces premiers résultats nous permettent de penser que si HDM est lancé dès le début du cycle de vie du site, il va pouvoir suivre l'évolution des fréquents rapidement, en acceptant les nouveaux items et en les intégrant immédiatement dans les population candidates. Nous avons également testé la qualité des résultats fournis par HDM, avec des fréquents de longueur 6 (Fig. 10.5) et des fréquents de longueur 12 (Fig. 10.6).

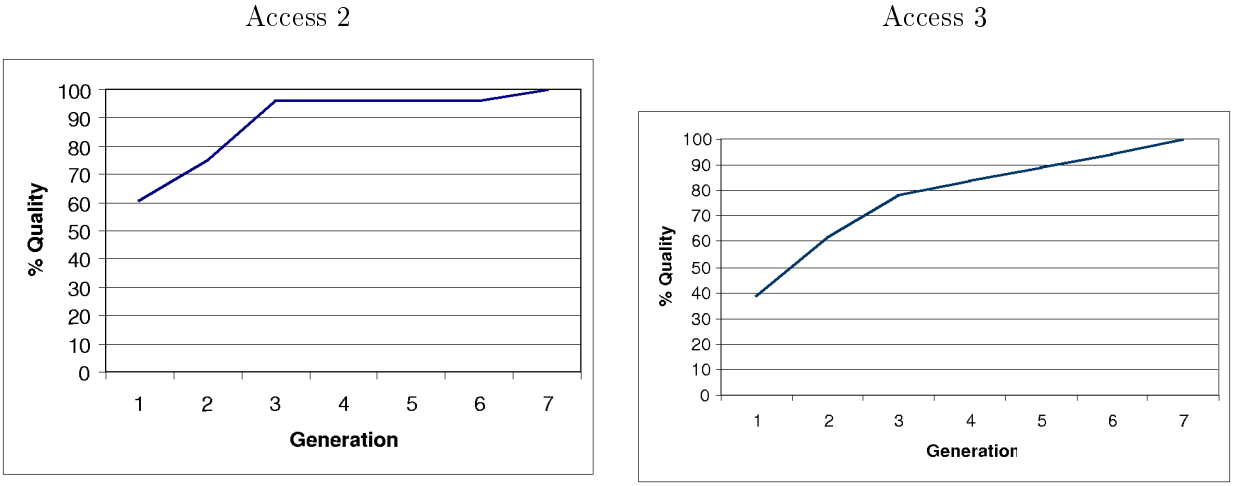

FIG. 10.5 – *Qualité des résultats pour les populations proposées*

La figure 10.5 montre, pour la base "Access 1", qu'à la troisième génération de candidats proposés par HDM, la qualité des résultats fournis est déjà de 96%. Il faut ensuite attendre la septième génération pour enfin atteindre les 100%.

Nous pouvons observer sur la figure 10.6 que, comme toute méta-heuristique, HDM accepte les régressions afin de ne pas être piégé dans un optimum local (de la génération 10 à la génération 11, la qualité passe de 67% à 63%).

10.4.3 Valider l'aspect temps réel de l'approche

Expérimentations sur un comportement de type évolutif :

Le but de ce test est de valider la capacité d'adaptation de l'algorithme HDM, en fonction de la qualité des résultats fournis pour un type de comportement qui en suit un autre. Pour cela nous avons utilisé deux bases de données (DB1 et DB2) qui recèlent des comportements bien distincts et que nous avons confiées à HDM. Dans un premier temps, seule DB1 est prise en compte, mais à chaque génération proposée par HDM, x% des séquences de DB1 sont remplacées par x% des séquences de DB2. Ce processus se réitère jusqu'à ce que DB1 soit totalement remplacée par DB2. Le test consiste alors à estimer dès le remplacement achevé, la qualité des résultats fournis pour DB2, en les comparant à ceux obtenus par PSP sur DB2. Les résultats de ces tests sont reportés à la figure 10.7.

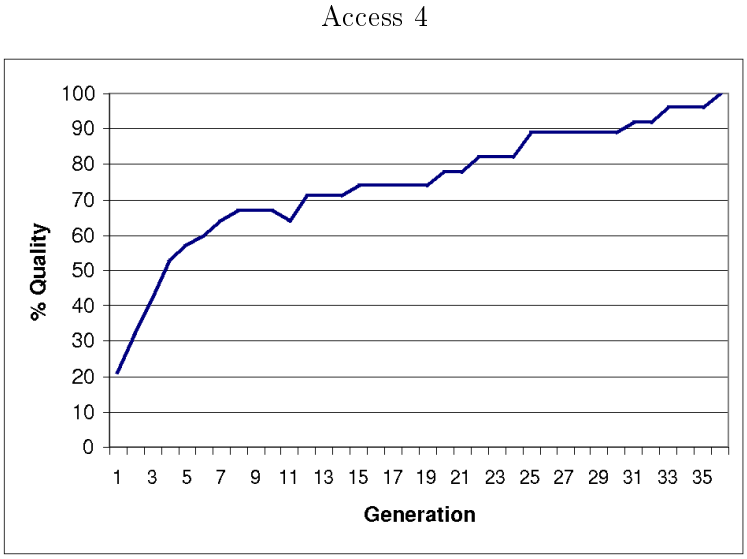

FIG. 10.6 – *Qualité des résultats pour les populations proposées, avec des fréquents de longueurs 12*

Ici, DB2 est respectivement : mushroom, chess puis access_log. Dans le cas des fichiers mushroom et chess, nous avons transformé les données qu'ils contennaient afin d'affecter à chaque item une transaction différente (donc une nouvelle date). Pour chaque fichier nous avons modifié x, le pourcentage de séquences dans DB1 remplacées par des séquences de DB2. Comme nous pouvions nous y attendre, plus le remplacement de DB1 par DB2 est rapide, moins les résultats proposés par HDM sont fiables par rapport à DB2 (par exemple pour le fichier chess, quand DB1 est remplacée au rythme de 1% par DB2, la qualité des résultat à la fin du remplacement total est de 100%, alors que si ce rythme est de 10%, la qualité passe à 53%). Cela est dû au fait que HDM n'a pas la possibilité de tester un nombre suffisant de populations sur DB2. En effet, un rythme de 10% signifie que HDM ne dispose que de 10 propositions et qu'en plus, le fichier DB2 ne lui est révélé qu'étape par étape. En revanche, nous pouvons constater qu'avec un remplacement assez progressif de DB1 vers DB2 (x=1% à chaque population), les résultats se situent entre 95 et 100%.

10.5 Discussion

La méthode présentée dans ce chapitre, propose des résultats qui affichent les avantages suivants :

Une disponibilité immédiate :

Etant calculés en temps réel, les résultats disponibles à un instant t reflètent, par nature, le comportement des utilisateurs connectés à ce même instant t. Les résultats obtenus peuvent alors être utilisés immédiatement de deux façons différentes :

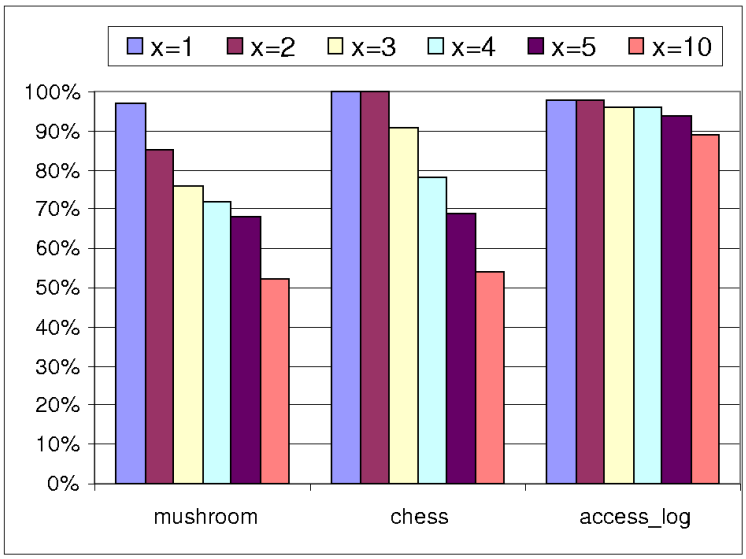

FIG. 10.7 – *Qualité des résultats pour la dernière population proposée par HDM, après un remplacement total de DB1 par DB2*

- En tant que résultats d'un processus de fouille de données, que l'on considère fiables et sur lesquels on peut faire reposer des décisions de type marketing ou modification hypertexte.
- En tant que pré-traitement (d'un coût quasiment nul), destiné à préparer le terrain pour un algorithme de fouille de données complet qui viendra s'appuyer sur les résultats obtenus afin d'orienter la génération de ses candidats.

Un nouveau type de séquences :

Les résultats obtenus permettent de répondre à la problématique de la recherche de motifs séquentiels avec une dimension supplémentaire : la possibilité d'historiser les résultats pour certaines "zones" de la base de données. La question généralement posée par un propriétaire de site web serait *"quel est le comportement fréquent de mes clients entre juin et août ?"*. Nous sommes alors effectivement dans le cadre d'un problème de fouille de données. Cependant, à l'heure actuelle, aucune méthode ne propose de répondre à la question *"Y a-t-il une période, pour laquelle ma base de données recèle plus de fréquents, et si oui laquelle ?"*. La réponse fournie peut alors afficher un décalage dans le temps par rapport aux prévisions de l'utilisateur, avec par exemple la "zone" suivante : *"La période du 12 Juin au 2 Septembre recèle un comportement fréquent à 72%"*. Ce comportement n'aurait peut-être pas été découvert, si l'on avait gardé la période demandée par l'utilisateur. Ces zones peuvent être découvertes en temps réel, au fil de la découverte de comportements par HDM, simplement en enregistrant les pics de fréquences, chaque pic correspondant alors à une zone d'intérêt, c'est à dire présentant les

caractéristiques suivantes :

- Au moins un motif fréquent présente un support particulièrement haut.
- La moyenne des supports des motifs fréquents de l'instant t est supérieure à celle de la population précédente et suivante (un "pic" du support moyen).
- Le nombre de motifs séquentiels dans la population est très haut (des comportements fréquents hétérogènes sont alors détectés) ou, au contraire, très bas.

Des résultats dédiés aux utilisateurs connectés :

Puisque les résultats sont obtenus en tant réel, l'utilisateur final (i.e. le propriétaire du site concerné) peut utiliser les résultats fournis de manière immédiate. Cela permet de cibler et d'envisager les comportements des utilisateurs actuellement connectés, grâce à l'analyse du comportement global des utilisateurs connectés au même instant.

Une puissance de calcul inépuisable :

Dans la mesure où HDM exploite la puissance de calcul disponible sur la machine de l'utilisateur connecté, le problème fournit lui-même la puissance nécessaire à le résoudre. Plus le problème s'étend, plus il apporte de puissance.

Fouille de données intéractive :

A tout moment, le serveur Java peut accepter un nouveau support utilisateur et l'envoyer à HDM immédiatement. HDM prendra alors en considération les modification avant d'analyser les résultats reccueilli chez les machines connectées et déterminer quelles séquences sont fréquentes. La prochaine population candidate sera alors proposée en fonction de ce nouveau support. Comme HDM fonctionne sans cesse, tant que le site est en ligne le support peut être modifié à tout moment.

Extraction de séquences de grande taille :

Dans la mesure où HDM ne fait pas une énumération exhaustive des sous-ensembles des motifs séquentiels fréquents, cet algorithme est très détaché de la taille des plus grandes séquences à découvrir (pour plus de détails concernant les relations entre la taille des fréquents et les temps de réponse voir section 12.2.2 ou les travaux décrits dans []). Nous pouvons donc affirmer que notre méthode peut trouver la plus grande partie (voire la totalité dans la majorité des cas) des motifs séquentiels, quelle que soit leur taille. Les expérimentations menées sur des bases de données locales, nous ont permis de valider cet aspect en utilisant HDM sur un fichier access log renfermant des motifs séquentiels dont la taille pouvait aller jusqu'à 40.

Quatrième partie

Conclusion et Perspectives

Chapitre 11

Conclusion

Pour conclure ce mémoire, nous proposons de faire un bref retour sur les objectifs, fixés en tant que cadre de travail par le chapitre exposant les problématiques. Pour chacune des problématiques, nous proposons un bilan des contributions, en soulignant les points forts de nos méthodes et algorithmes, ainsi que leur adéquation avec les objectifs fixés. Les applications que nous avons implémentées sont ensuite évoquées, avant de faire un retour vers le processus d'ECD, dont l'étape de prétraitement se prête aux améliorations que nos contributions laissent entrevoir.

Nous avons vu, dans la discussion sur l'état de l'art, que l'arbre de hachage, destiné à l'extraction de motifs séquentiels, pourrait tirer profit d'une mise en facteur des préfixes des candidats qui tienne compte des changements d'itemsets. Cette factorisation, implémentée par la structure d'arbre préfixée que nous proposons, et son exploitation par notre algorithme PSP, permettent effectivement de gagner à la fois de l'espace en mémoire mais aussi du temps à l'exécution de l'algorithme de fouille de données. Nos expérimentations ont montré que PSP pouvait extraire des motifs séquentiels avec des temps de réponse jusqu'à 5 fois inférieurs à ceux du parcours de l'arbre de hachage. La force principale de PSP réside en deux points :

- Les changements d'itemsets sont maintenus par l'arbre de hachage. Cela revient à dire que lors d'un parcours de l'arbre, l'algorithme sait vers quels items aller, selon qu'ils appartiennent ou pas à l'itemset en cours. Cette caractéristique permet de ne pas re-tester un ensemble de candidats dont on vient de faire le parcours, comme le ferait un parcours de l'arbre de hachage.
- Les optimisations sur le parcours. Nous avons mis en place des techniques qui évitent, pendant le parcours, d'examiner plusieurs fois la même branche de l'arbre avec le même candidat. Ces optimisations, considérées dans leur ensemble, sont un atout qui permet à PSP d'offrir les temps de réponse décrits.

Le problème rencontré par GSP, le premier algorithme qui fut proposé pour résoudre les contraintes de temps, se situe au niveau du calcul de ces contraintes à la volée, pendant le parcours de l'arbre. Nous avons donc mis en place un graphe, destiné à représenter toutes les combinaisons possibles d'une séquence de données, en fonction des contraintes de temps. Ce graphe, calculé par l'algorithme GTC, est déterminé avant le parcours de l'arbre et les contraintes de temps n'ont alors plus à être évaluées.

Les avantages de ce graphe sont principalement :

- Calculer une fois pour toutes les combinaisons. Une fois ce calcul effectué, alors les possibilités offertes par la séquence de données avec les contraintes de temps spécifiées sont utilisées pour détecter les candidats inclus et incrémenter leur support. Cette méthode a pour principal avantage d'éviter les, trop nombreux, retour-arrières ("back-tracking") employés par GSP. Nos expérimentations sur ce point on, en effet, montré une corrélation directe entre les temps de réponse et le nombre d'appels récursifs au parcours de l'arbre.
- Détecter des inclusions parmi les combinaisons possibles. Nous avons, en effet, constaté que parmi les combinaisons, dues aux contraintes de temps, certaines étaient incluses dans d'autres, plus générales. Considérons une combinaison d_1 incluse dans une combinaison d_2, issue de la même séquence de données d. Si un candidat c est supporté par d_1 alors il est supporté par d_2. De plus si c est supporté par d_2 alors il est supporté par d. Donc il est inutile de tester d_1 pour déterminer si c est supporté par d, seul le test avec d_2 est utile.

Les algorithmes de recherche incrémentale de motifs séquentiels, que nous avons vu dans l'état de l'art, présentent tous le même problème face à la taille de la base de données. Que cela soit en raison d'une ré-écriture de la base de données, ou bien à cause de son chargement en mémoire, la taille de cette dernière est un obstacle au bon déroulement de ces méthodes. La méthode générer-élaguer propose justement de ne dépendre, en espace mémoire, que de la taille du résultat. C'est pourquoi nous avons orienté nos travaux vers une extension de cette méthode, afin de garder cette caractéristique. L'algorithme ISE, que nous avons mis au point, propose de calculer les extensions qui seront testées dans l'étape suivante. Les extensions sont générées puis testées en tant que séquences à part entière. Si elles sont fréquentes, alors elles peuvent être testées en tant qu'extension d'autres séquences. Lors de ce test, les extensions de la prochaine étape sont déjà testées en tant que séquences. Ce pas de décalage dans la méthode générer-élaguer incrémentale, nous a permis de réduire considérablement le nombre de candidats à vérifier sur la base de données, et donc les temps d'exécution. De plus nous avons mis en place des tests visant à tester la capacité d'ISE en dehors du contexte incrémental. Le but était de comparer les temps d'exécution de deux méthodes. La première méthode comportait 3 étapes : découper U en DB+db, utiliser GSP sur DB et enfin utiliser ISE sur db pour obtenir les résultats. Soit T_{ISE} le temps cumulé de ces trois étapes. La seconde méthode ne comportait qu'une seule étape : utiliser GSP sur U. Soit T_{GSP} le temps d'exécution relevé pour la seconde méthode. Nous avons pu remarquer que selon le découpage qui était fait de U (en DB+db), T_{ISE} pouvait être inférieur à T_{GSP}. Nous avons alors mené des expérimentations pour déterminer les cas de figure favorables à ce phénomène.

Pour terminer la première partie de notre contribution, nous avons présenté deux types d'applications : le Text Mining et le Schema Mining. Si l'analogie entre la recherche de structures de phrases communes à un ensemble de textes et la recherche de motifs séquentiels semble assez accessible, il n'en est pas de même pour la recherche de sous-schémas communs à un ensemble de schémas. Cette seconde analyse demande une transformation des données assez précise et se fait via plusieurs étapes. La première consiste à transformer les sommets des arbres à traiter en items. La seconde procède à un parcours préfixé de l'arbre transformé pour en obtenir la séquence. Ensuite un algorithme de fouille de données peut-être appliqué sur les séquences obtenues et la transformation inverse des séquences fréquentes permet d'obtenir une structure commune aux schémas analysés. Les expériences que nous avons réalisées, nous ont permis de constater la cohérence de cette approche, grâce à une vérification

du support des structures proposées par notre architecture.

La seconde partie de notre contribution s'articule autour de problèmes liés à l'analyse du comportement des utilisateurs d'un site web. Après avoir défini et implémenté l'architecture WebTool, nous avons proposé deux problématiques destinées à augmenter la connaissance issue d'un processus de Web Usage Mining. Pour parvenir à ce but, nous avons fixé les caractéristiques que doivent proposer, selon nous, les nouvelles méthodes de Web Usage Mining :

Plus d'informations sur les comportements :

- *Grâce à l'inter-sites.* L'algorithme WUMIS, que nous avons proposé dans le cadre du Web Usage Mining Inter-Sites, propose d'analyser le comportement des utilisateurs en tenant compte de leur historique. Cet historique, fournit par le site ayant redirigé l'utilisateur sur le site concerné, permet de classer l'utilisateur en fonction de son passé, dans des catégories plus précises. Les comportements obtenus sont ensuite calculés en fonction des seuls utilisateurs qui appartiennent à cette catégorie (celle des utilisateurs qui viennent de ce même site). Le support, dans cette nouvelle catégorie, permet alors d'obtenir des fréquents dotés d'une signification plus élevée.
- *Grâce au temps réel.* L'architecture HDM, qui permet d'extraire les comportements des utilisateurs d'un site web en temps réel, ouvre des possibilités nouvelles sur les connaissances obtenues. En effet, par sa nature d'outil temps réel, HDM propose des comportements qui sont issus de l'analyse instantanée des utilisateurs connectés. Les fréquents obtenus sont alors dédiés aux utilisateurs connectés et non plus à tous les utilisateurs confondus du site, sur une période trop étalée dans le temps. Nous avons également montré que HDM permet l'extraction de longs motifs séquentiels, la découverte de périodes d'intérêt supérieur (taux de fréquents très fort ou très faible, ou bien des "super-fréquents" avec un support très élevé) ou encore la fouille de données intéractive.

Plus d'adaptation de la part du système :

- *Grâce à l'inter-sites.* Nous avons vu le double avantage, côté système, du Web Usage Mining Inter-Sites. Le premier vient de la rapidité de réaction que permettent les résultats obtenus. En effet, tenir compte de l'historique de navigation d'un utilisateur permet de faire des prédictions immédiatement après son arrivée. Dès la première page du site demandée, le système peut, en fonction des actions passées de cet utilisateur, prédire son comportement et proposer des liens hypertexte. Le second avantage provient de la richesse des comportements obtenus. Dans ce contexte, le système dispose de plus d'informations pour ses prédictions. Sa marge de manœuvre est alors plus élevée.
- *Grâce au temps réel.* Avec HDM, l'information dont dispose le système n'est plus "Les comportements fréquents des utilisateurs du mois dernier, toutes connexions confondues, sont : <...>" mais plutôt "Les comportements fréquents des utilisateurs *actuellement connectés* sont : <...>". Cette différence permet au système d'adaptation dynamique des pages du site de gagner en performance et en pertinence. En effet, pour s'adapter à un utilisateur u, nous pensons que la meilleure information à analyser reste celle qui concerne les utilisateurs connectés en même temps que u. Cela permet une classification naturelle des utilisateurs, en fonction de leur période de connection et donc une adaptation accrue de la part du système de modification dynamique des pages.

Le nombre d'applications, sur lesquelles nous avons été conduits à travailler, nous a fait prendre conscience d'une étape cruciale dans le processus d'ECD. Cette étape est le prétraitement des données. Nous avons réalisé son importance en spécifiant, puis en implémentant, la plateforme de fouille de données *L*eitmotiV. L'objectif de *L*eitmotiV est de regrouper le plus grand nombre d'algorithmes de fouille de données, mais aussi d'applications dérivées de ces algorithmes. Parmi ces applications, nous pouvons compter, par exemple, l'analyse d'un fichier access log ou encore le Text Mining. Nous avons réalisé que *L*eitmotiV prenait une ampleur de plus en plus proche de celle d'un processus d'ECD. En effet, à chaque application, notre plateforme associe les étapes suivantes :

- Prétraitement des données. Cette étape étant réalisée par un parser, écrit par nos soins et destiné à transformer les données du problème d'origine en données compréhensibles par un algorithme de fouille de données.
- Fouille de données.
- Transformation des résultats. L'objectif de cette étape étant de traduire les résultats, fournis par l'algorithme de fouille de données, en résultats correspondants à la problématique d'origine.

La qualité et la pertinence des résultats obtenus dépendent de l'étape de prétraitement, tout autant que de l'algorithme de fouille de données. Cependant, nous avons constaté que l'analyse des données à traduire, qui se solde par l'écriture du parser, méritait une étude attentive. En effet, ces deux étapes pourraient, à l'aide d'un processus de découverte de structures fréquentes, être automatisées ou semi-automatisées. Cette amélioration de l'étape du prétraitement, dans le processus d'ECD, est envisagée parmi les perspectives que nous décrivons dans le chapitre suivant.

Chapitre 12

Perspectives

L'efficacité des algorithmes, issus des contributions de la communauté internationale, n'a cessé d'être améliorée jusqu'à ce jour. La fouille de données peut désormais se confronter à d'autres domaines, afin de tirer profit de techniques venues de disciplines qui ont acquis leur lettre de noblesse.
Les perspectives ouvertes par les travaux présentés dans ce mémoire sont liées, par exemple, aux possibilités que peuvent apporter des domaines comme le Langage Naturel ou les techniques de génération de code source et de compilation. Elles concernent aussi les améliorations que l'on peut apporter aux résultats fournis, en terme de traitement *A-posteriori*.
Nous développons, dans la section 12.1, deux perspectives directement liées à l'extraction de motifs séquentiels. Ces techniques sont issues de contraintes, que peuvent apporter certains contextes de travail, et de l'analyse des avantages que l'on peut trouver à les prendre en compte. La section 12.2, quant à elle, propose une réflexion que nous menons à l'heure actuelle sur le prétraitement des données. Si ce domaine est, à ce jour, encore envisagé au cas par cas, il présente des caractéristiques qui laissent présager la possibilité d'une automatisation de certains points dans la transformation des données.

12.1 Perspectives générales

12.1.1 Vers une approche efficace pour le Text Mining

L'application de techniques de recherche de motifs séquentiels sur des documents a montré qu'il était envisageable de déterminer quels sont les éléments d'un texte qui sont fortement corrélés. En outre, la prise en compte des contraintes de temps adaptées aux composants d'un texte (chapitre, section, paragraphe, mot, etc.) offre de nouvelles possibilités d'analyse de texte sur la globalité des documents. Par exemple, la contrainte de minGap peut être utilisée pour préciser que nous ne sommes intéressés que par des régularités entre phrases, alors que la contrainte de windowSize va permettre de regrouper les phrases entre elles. Les expériences menées sur des pages Web extraites ainsi que sur un très grand nombre de fichiers pdf ont montré que les résultats obtenus pouvaient permettre par exemple de classifier les textes en fonction des occurrences des termes. Une autre application possible de ces analyses est la possibilité de rechercher des tendances dans des documents et a été proposée dans [LAS97].

Le principe général suivi pour l'analyse de fichier texte repose tout d'abord sur un pré-traitement particulier des données. En effet, lors de ce pré-traitement il est indispensable de supprimer les termes jugés non significatifs comme les articles tout en tenant compte de leur position dans le texte. La gestion d'un thésaurus est également nécessaire pour faciliter la découverte de corrélation. Même si les motifs séquentiels sont adaptés ils sont malheureusement limités à une analyse syntaxique des documents et une perspective intéressante est alors de compléter cette analyse par une analyse sémantique. Une collaboration avec l'équipe Langage Naturel du Lirmm débute sur cette perspective. Le principe général est d'étendre la recherche de motifs à la prise en compte de vecteurs sémantiques associés aux paragraphes ou à d'autres composants du document. En combinant ces deux aspects nous espérons être capable de déterminer non seulement les occurrences répétées mais également les éventuelles répétitions de parties de documents de sémantiques proches.
Intégrer une composante vecteur sémantique aux motifs séquentiels nécessite cependant de prendre en compte pour chaque item participant, i.e. pour chaque mot du document, quel est son vecteur sémantique le plus proche. Ce paramètre complique alors l'extraction des motifs fréquents dans la mesure où les éléments à analyser ne sont pas de même type.

Associée à la sémantique du document, une composante complémentaire d'analyse peut être envisagée en considérant les travaux que nous avons menés sur la recherche de régularités de structures complexes. En se basant sur une analyse de la structure syntaxique du document et en recherchant au travers des documents les structures analogues nous pouvons alors obtenir une information supplémentaire qu'il semble intéressant de combiner avec les résultats de l'analyse précédente de manière à affiner la phase de recherche de connaissances.

12.1.2 Vers une temporalité accrue des résultats

Les résultats obtenus par les algorithmes d'extraction classiques ne sont pas suffisamment précis si l'on souhaite analyser de manière détaillée le comportement des utilisateurs ou des clients au cours du temps. Par exemple, il est possible de connaître en analysant une base de données de transactions que *62% des clients achètent un plat cuisiné puis un livre de recettes.* De la même manière, en tenant compte des contraintes de temps, nous pouvons déterminer que *60 % des clients qui ont visité l'URL* `/jdk1.1.6/docs/api/Package-java.io.html` *et l'URL* `/jdk1.1.6/docs/api/java.io.BufferedWriter.html`*, ont également visité, dans les 30 jours suivants, l'URL* `/jdk1.1.6/docs/relnotes/deprecatedlist.html` (voir section 8.1). Même si ces informations sont pertinentes elles ne précisent pas la répartition dans le temps des clients qui participent aux support. Il est beaucoup plus intéressant de savoir que sur les 62% des clients, *3% des clients achètent un plat cuisiné puis un livre de recettes dans les 3 jours et 59% des clients achètent un plat cuisiné puis un livre de recettes au moins 20 jours après.* En effet, en utilisant uniquement le support minimal nous ne pouvons déterminer qu'un comportement très général. Une première solution triviale serait dans notre exemple de spécifier un minGap de 20 jours. Malheureusement cette solution a plusieurs inconvénients majeurs : (i) les 3% des clients ne vérifiant pas la contrainte de minGap ne sont pas extraits et si le support minimal a été spécifié à 62%, nous n'obtenons pas non plus les 59% des clients de la base ; (ii) La spécification des contraintes de temps est délicate et nécessite généralement une première extraction

qui pourra être affinée par la suite.

Nous avons mené une première étude qui a montré qu'il était possible de générer à l'aide d'une nouvelle passe sur la base les différentes répartitions possibles des clients par rapport aux motifs fréquents trouvés. Le principe général est de stocker à la fois le motif séquentiel et un arbre préfixé. Lors du parcours sur la base de données, si la séquence du client est incluse dans le motif séquentiel, celui-ci est inséré dans l'arbre préfixé en précisant les délais entre chaque itemset ainsi que le nombre de clients vérifiant ce motif. Si une nouvelle séquence est incluse dans le motif mais avec un délai entre itemset différent, celui-ci est inséré via un nouveau lien pour préciser qu'il s'agit d'un autre délai et le compteur associé au nouveau nœud cible de ce lien est incrémenté. Un simple parcours des arbres générés offre alors la possibilité d'offrir à l'utilisateur la répartition des différents clients par rapport aux temps séparant les itemsets. Si cette approche semble bien adaptée, l'une des conséquences immédiate est la génération d'un trop grand nombre de répartitions rendant inexploitables les résultats de l'extraction. Dans notre exemple, une telle approche aurait pour conséquence de répartir les 59% des clients de la base en autant de délais entre itemsets supérieurs à 20.
Une étude plus approfondie est donc indispensable pour générer les intervalles de temps les plus significatifs éventuellement en les intégrant directement dans l'algorithme de recherche de motifs. Les travaux menés dans le cadre de la recherche de règles d'associations quantitatives [SA96a], même s'ils sont définis dans un autre contexte, pourrait être un point de départ à la définition de tels intervalles. En outre, adaptés aux données de la base, les principes issus d'une telle recherche semblent offrir une solution plus générale à la définition d'une approche de recherche de motifs séquentiels quantitatifs. En effet, les algorithmes de règles quantitatives ne pouvant pas être adaptés aux motifs, une approche spécifique est indispensable [Sri96] et il n'existe pas aujourd'hui de solution à la recherche de tels motifs qui reste toujours un domaine de recherche ouvert [Sri96, Zak98].

12.2 Intégration dans le processus de KDD

Dans le chapitre 7, nous avons proposé des exemples d'applications pour lesquelles les motifs séquentiels sont particulièrement adaptés. Cette adaptation peut se traduire par une transformation simple des données d'origine du problème à traiter en instance d'un problème de motifs séquentiels, ou bien par une efficacité de l'application d'algorithmes d'extraction de motifs séquentiels sur des données ayant subi une transformation non triviale (comme le Schema Mining par exemple). Parmi les transformations les plus simples, nous pouvons citer le Text Mining, le Web Usage Mining ou encore les transformations de données venant de groupes industriels avec lesquels nous avons été conduits à travailler.

Le nombre croissant d'applications pour lesquelles nous avons développé nos effort en vue d'acquérir, transformer, analyser puis présenter les résultats d'une étape de fouille de données, nous a tout naturellement incité à produire une plateforme d'accueil pour les algorithmes concernés.

Ce chapitre a pour but de présenter deux aspects, fortement corrélés, de ces travaux. Tout d'abord, nous proposons un tour d'horizon des fonctionnalités requises de la part d'une plateforme de fouille de données acceptant des sources de données hétérogènes. Ensuite nous proposons des travaux menés dans

le cadre de la transformation des données, qui visent une automatisation générique de cette étape en prenant un niveau d'abstraction supplémentaire (les détails relatifs à ces travaux peuvent être trouvés dans [Jai01]).

12.2.1 Accueil des composantes de fouille de données

Afin de regrouper les algorithmes développés dans le cadre de ce mémoire, nous avons mis au point une structure d'accueil pour les différentes étapes que suit un processus d'ECD classique. Le but de cette structure étant également d'aider l'utilisateur à organiser l'enchaînement des étapes, à saisir les paramètres des différents algorithmes et à traiter les résultats. Cette plateforme, appelée *L*eitmotiV, est à l'heure actuelle capable de traiter les étapes suivantes :

- **transformation** des données à l'aide d'un parser existant,
- acquisition des **paramètres** de l'étape fouille de données,
- application d'algorithmes de **fouille** de données,
- transformation inverse des résultats afin d'obtenir une **sortie lisible** par l'humain, grâce à un deparser,
- **manipulation**, stockage et interrogation des résultats.

Une phase de subsitution, destinée à remplacer la transformation des données, fait à l'heure actuelle l'objet de travaux au sein de l'équipe Data Mining du Lirmm. Cette étape consiste à automatiser la traduction des données en proposant de composer elle- même le parser à employer, plutôt que d'utiliser un parser existant. Cet aspect est plus amplement développé dans la section 12.2.2 et dans [Jai01].

A l'heure actuelle *L*eitmotiv est capable de traiter des données provenant de :

- série de documents **textuels**,
- fichiers **log** de serveurs web Apache et IIS,
- schémas provenant de l'**imdb**.

L'exemple 52 illustre le comportement de *L*eitmotiV dans le cadre de l'extraction de comportements fréquents des utilisateurs d'un site Web.

Exemple 52 *Tout d'abord, l'utilisateur doit spécifier l'emplacement du fichier log à analyser. Cet emplacement lui est demandé comme illustré dans la figure 12.1. Une fois le projet créé, la plateforme transforme les données pour obtenir un format interprétable par le moteur de fouille de données. L'utilisateur est alors amené à spécifier les paramètres avec lesquels la phase de Data Mining sera exécutée (figure 12.2). Le processus de fouille de données est alors amorçé, avec en cours d'exécution des rappels sur les paramètres et des informations sur l'évolution de la fouille (figure 12.3). Une fois les résultats obtenus et transformés en séquences lisibles, l'utilisateur peut manipuler ou interroger la réponse qui lui est fournie (figure 12.4).*

Cependant, cette plateforme, même si elle offre une intégration des modules développés dans le cadre des travaux présentés dans ce document, reste une façon d'implémenter le processus d'ECD classique. Cela suppose donc l'existence d'un processus écrit et fonctionnel de transformation des données pour

chaque type d'entrée que cette structure propose de traiter. Pour échapper à cette contrainte et proposer une transformation adaptée, quel que soit le type de données en entrée, des fichiers à analyser, nous avons menés des travaux destinés à fournir un moyen de comprendre la structure d'un fichier sans la connaître *a-priori*. La compréhension de cette structure permettant alors de transformer les données sans avoir à écrire le parser serait le but de la version idéale de cette étape. La section 12.2.2 offre un tour d'horizon des techniques mises au point dans ce contexte.

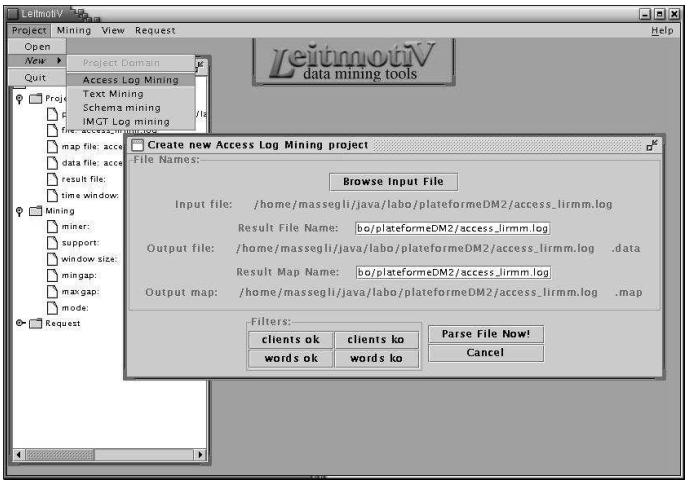

FIG. 12.1 – *acquisition du fichier log à traiter*

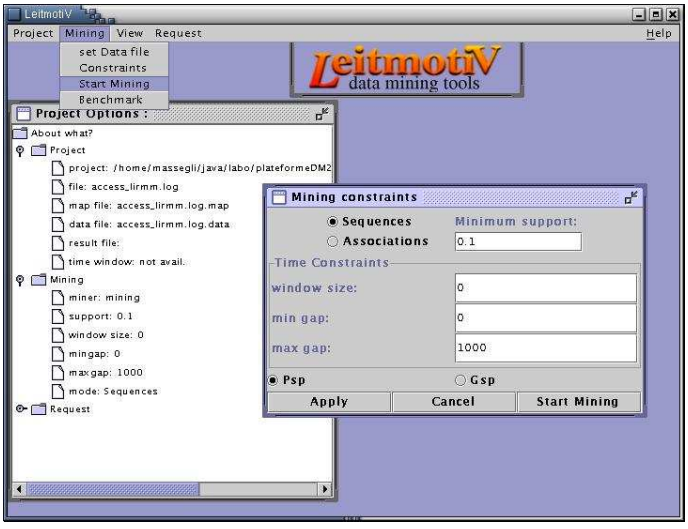

FIG. 12.2 – *Paramètres avec lesquels procéder à l'étape de fouille de données*

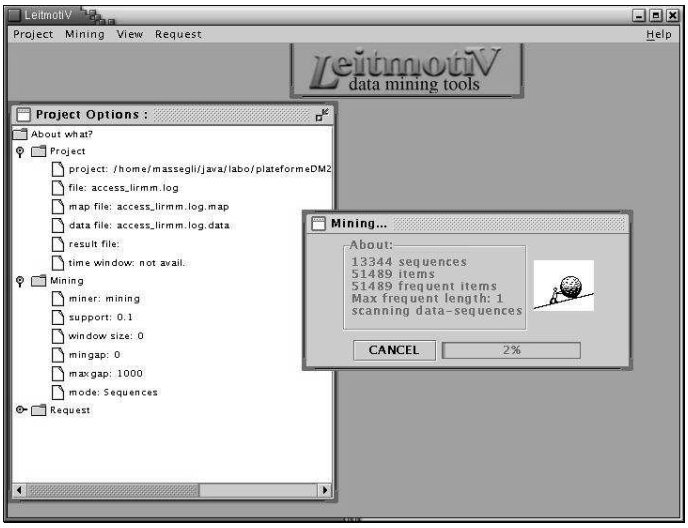

FIG. 12.3 – *Processus de fouille de données en cours d'exécution*

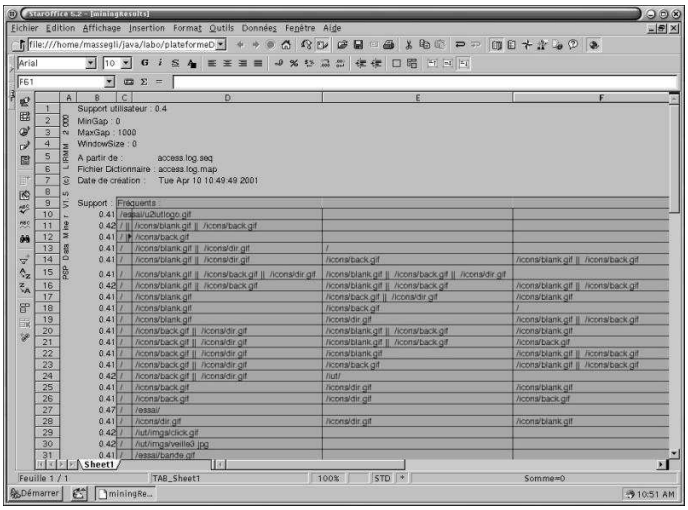

FIG. 12.4 – *Affichage des résultats interprétables*

12.2.2 Réflexions sur le pré-traitement des données

Dans notre contexte, l'étape de fouille de données correspond à l'application stricte d'un algorithme, qui s'efforce d'être le plus rapide possible, pour résoudre une problématique fixée : l'extraction de motifs séquentiels. Comme nous l'avons vu en introduction de ce mémoire, la fouille de données est une étape de ce processus plus global qu'est l'ECD. Cette section propose d'analyser, puis s'efforce d'améliorer, une étape située en amont de ce processus : le pré-traitement des données. Nous verrons, dans cette

Entrée	N^o CB	Banque	Accord	Date	Article	Fournisseur	Quantité
1	×	×	×	×	×		×
2	×		×	×	×	×	×
3	×	×	×	×	×	×	
4	×	×		×	×	×	×

FIG. 12.5 – *Champs renseignés dans le fichier du cas 1*

N^o CB	Date	Article
0134 596 83	01/07/2001	soda
9458 875 32	02/07/2001	housses
⋮	⋮	⋮

FIG. 12.6 – *Les champs retenus pour l'étape de transformation*

section, le déroulement de quelques processus classiques de transformation des données, puis nous étudierons les points communs de ces transformations. A partir de cette observation, nous analyserons les possibilités d'automatiser cette transformation grâce aux travaux proposés au sein de l'équipe Data Mining du Lirmm.

Transformation classique

Considérons trois cas d'applications des motifs séquentiels pour lesquels la transformation des données est relativement directe :

- Cas 1 : un fichier fourni par une boutique de station essence, qui vend divers articles sur l'autoroute. Ce fichier résume les achats des clients dans cette boutique.
- Cas 2 : un fichier access log, venant d'un site web qui désire analyser le comportement de ses utilisateurs.
- Cas 3 : un fichier de relevé des interventions dans une raffinerie. Ce fichier résume les réparations effectuées chaque jour sur les appareils de la compagnie.

Cas 1

Le fichier à analyser contient des informations diverses, mais pas forcément renseignées, sur les achats effectués dans la boutique concernée. Ces informations sont le N^o de la carte bancaire avec laquelle est effectué l'achat, la banque à laquelle est enregistrée cette carte, un numéro d'accord de prélèvement obtenu auprès de la banque (s'il est obtenu), la date de l'achat, la désignation de l'article, le fournisseur de cet article et la quantité d'articles achetés. La figure 12.5 illustre un exemple de fichier venant de cette boutique, réduit à l'information "le champ est renseigné ou pas" (si une croix apparaît pour l'entrée X et le champ Y alors le champ Y est renseigné pour cette entrée).

Nous savons que pour une recherche de motifs séquentiels, les champs qui seront intéressants sont : le numéro de carte, la date de l'achat et l'article. L'étape de transformation va donc commencer par filtrer les données afin d'obtenir des entrées du type décrit par la figure 12.6.

Intervention	Description Pb	Technicien	N^o machine	Demandeur	Résultat	Date	Durée
1	×	×	×		×	×	×
2	×	×	×	×	×	×	×
3	×	×	×		×	×	×
⋮	×	×	×	×	×	×	×

FIG. 12.7 – *Champs renseignés dans le fichier du cas 3*

Il suffit, une fois les données filtrées, d'associer à chaque numéro de carte bancaire un numéro de client, à chaque date un numéro de date et chaque article un numéro d'article. Le fichier résultant de cette transformation peut alors être analysé par un algorithme de fouille de données.

Cas 2

Le cas 2 est un cas particulièrement bien adapté à la problématique des motifs séquentiels. Dans ce cas, le fichier log contient diverses informations comme le numéro IP du client, le type de connection, la date de connection, l'URL demandée, le protocole, le sens de communication. Ces champs ne sont pas toujours remplis et le propriétaire du site Web sait ce qu'il veut : connaître les enchainements d'URL fréquents chez ses clients. Ce cas est alors résolu en filtrant le fichier en entrée pour ne garder que le numéro IP, la date et l'URL. À chaque numéro IP est alors associé un numéro de client, à chaque date un numéro de date et à chaque URL un numéro d'item.

Cas 3

Dans le cas de la raffinerie, le fichier contient des relevés d'interventions sur des machines appartenant à une compagnie pétrolière. Les relevés, identifiés par un numéro d'intervention et une description du problème, concernent le technicien, le type de son intervention, le numéro de machine réparée, le demandeur de la réparation, le résultat de la réparation, la date d'intervention et la durée de cette intervention. Ces champs ne sont pas forcément toujours renseignés (sauf la description du problème, qui peut être choisie parmi un catalogue de descriptions disponibles) et la compagnie voudrait connaître les possibilités offertes par une analyse de ce fichier.

Pour connaître ces possibilités, il faut tout d'abord procéder à une observation des champs présents dans la structure, comme l'illustre la figure 12.7.

L'intérêt de ce cas porte sur le nombre de champs qui sont renseignés pour toutes les entrées. Cela confère un grand nombre de possibilités quant aux résultats que l'on peut obtenir et nous permet d'illustrer combien le prétraitement des données s'avère capital. L'entreprise en possession d'un tel fichier peut en effet obtenir, après une étape de fouille de données, des résultats variés, en fonction de la façon dont le fichier est transformé pour être ensuite traité par l'algorithme de fouille de données. Ces résultats peuvent être :

– Question 1 : "Y a-t-il un ordonnancement des réparations d'une machine à l'autre ?". En d'autres

termes peut-on établir que la machine x est fréquemment en panne avant la machine y et que l'intervention sur x est suivie d'une intervention sur y puis sur z, etc.

- Question 2 : "Quels sont les techniciens dont les interventions sont les plus corrélées ?". Il s'agit là de trouver un enchaînement fréquent du type "le technicien A intervient avant et après le technicien B dans 80% des cas.
- Question 3 : "Y a-t-il une succession de réussites ou d'échecs, en fonction des machines réparées ?". Dans ce cas la réponse peut être du type "La machine x subi une réparation réussie puis trois réparations ratées. Ensuite la machine z subi une réparation réussie et enfin la machine x subi une réparation réussie". Admettons que cette séquence se retrouve dans 65 entrées sur 100. Un tel résultat peut permettre d'établir un lien entre l'échec des réparations sur x et le fait que z nécessitait une réparation. Une fois z réparée, alors x est réparée avec succès dans 65% des cas.
- Question 4 : "Y a-t-il une succession de reussites ou d'échecs, en fonction du technicien ?". Cette question, qui du point de vue de l'éthique peut paraître "dangereuse", trouve sa réponse dans des séquences du type "Le technicien A effectue deux interventions avec succès, puis le technicien C effectue une intervention avec succès".

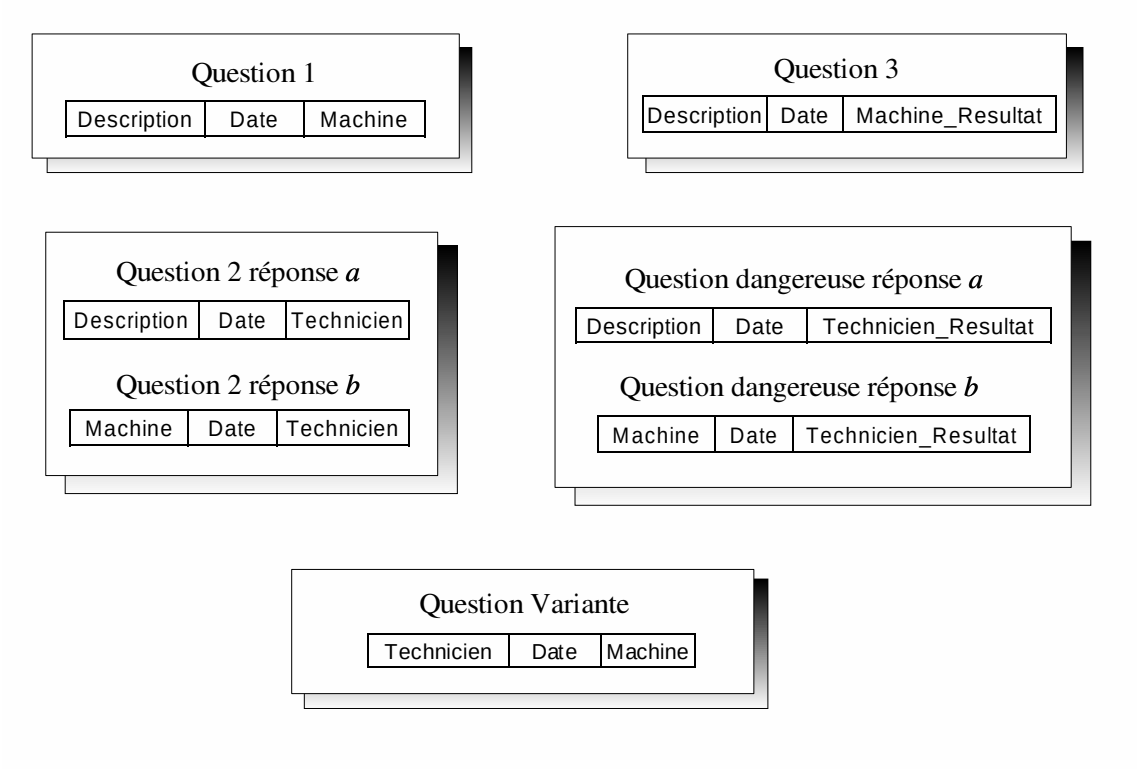

FIG. 12.8 – *Les réponses aux différentes questions du cas 3*

Nous nous limiterons à ces quelques exemples car le nombre de possibilités est assez grand. Chacune de ces questions nécessite une phase de transformation des données différente, suivie d'une étape de fouille de données qui lui est consacrée. La figure 12.8 illustre les transformations nécessaires pour répondre

à chacune de ces questions.
La question 1 trouve une réponse grâce à la transformation suivante : associer à chaque description un numéro de client, à chaque date un numéro de date et à chaque machine un numéro d'item. Le processus de fouille de données sera alors en mesure de sortir des séquences du type : $<$(machine x) (machine y) (machine z) $>$. L'enchaînement de réparations x puis y puis z est alors un enchaînement fréquent.
La question 2 est résolue par l'application de la transformation suivante : associer à chaque description un numéro de client, à chaque date un numéro de date et à chaque technicien un numéro d'item. De la même façon que pour la question 1, mais cette fois concernant les techniciens, on peut trouver des enchaînements fréquents. Une variante de cette réponse consiste à associer le numéro de client, non plus à la description, mais à la machine. Ce changement permet d'obtenir des enchaînements plus précis, puisqu'il s'agit de trouver les successions de techniciens sur une machine (quelle que soit cette machine). La première réponse permettait de trouver ces successions toutes machines confondues.
Pour répondre à la question 3, il faut en revanche pousser la réflexion un peu plus loin et procéder à la transformation suivante : associer à chaque description un numéro de client, à chaque date un numéro de date et ensuite chaque couple [machine,résultat] sera concaténé en un seul mot pour être ensuite transformé en numéro d'item. Ainsi le processus d'extraction de motifs séquentiels sera capable de fournir des résultats du type : $<$(machine a - succès) (machine a - échec) (machine a - échec) (machine a - échec) (machine b - succès) (machine a - succès) $>$.
Enfin, la question 4 n'est autre que la question 2, avec une considération portée sur le technicien à la place de la machine et une concaténation avec "Résultat" pour reprendre le principe de la réponse 3. Il suffit donc de remplacer "Machine" par le couple [technicien,résultat] pour connaître la réponse aux questions posées, ou pour comprendre leur sens. Le type de séquences fournies sera donc, par exemple, $<$(Tech$A$ - succès) (Tech$A$ - succès) (Tech$C$ - succès) $>$.
Afin de constater les possibilités qui s'offrent, à l'analyse d'un tel fichier, considérons la question suivante ("question variante" de la figure 12.8) : "De manière générale les techniciens interviennent-ils sur les machines dans un ordre particulier ?". La réponse à cette question trouve son originalité dans le fait que le numéro de client n'est pas associé à la description mais au numéro de technicien. Le type de séquences que l'on peut trouver sera donc : $<$(machine a) (machine y) (machine a) $>$, ce qui peut permettre de conclure sur un enchaînement particulier des interventions, tous techniciens confondus et toutes machines confondues.

Nous venons de voir 3 cas de fichiers en entrées pour un processus d'ECD. Dans un processus d'ECD classique, on considère que l'utilisateur connaît la transformation à appliquer, qu'elle soit relativement directe comme pour le cas 2 ou plus délicate comme pour le cas 3. Cependant, il est intéressant de noter un point commun à toutes ces transformations : les champs que l'on conserve pour le processus de fouille de données sont les champs ayant un fort taux de présence dans le fichier. Nous en déduisons qu'il est possible de déterminer la structure (au sens "`client - date - item`") du fichier à analyser en retenant les répétitions d'un enregistrement sur l'autre. Observons par exemple le fichier du cas 1. Nous y voyons une répétition de l'information "champ renseigné" pour les champs "N^o CB", "date" et "article". Cette répétition permet de penser que ces trois champs sont à retenir pour une phase de fouille de données. Cette observation, que l'on peut confier à un algorithme de fouille de données, peut alors aider de manière plus ou moins forte, à tranformer le fichier en une instance du problème de la recherche de motifs séquentiels.

Transformation automatique

Le processus d'ECD que nous avons vu en introduction admet une phase de transformation déjà établie et que l'on va appliquer sur les données. Nous proposons un nouveau processus d'ECD, dans lequel la phase de transformation est remplacée par une phase d'analyse des données, recherche de structure, mise au point du filtre (de transformation) et enfin traduction. Dans le processus de génération du traducteur, illustré par la figure 12.9 (dans le cadre en pointillés), il est possible de revenir de n'importe quelle étape vers une des étapes précédentes. Le reste du processus suit le comportement classique d'un processus d'ECD.

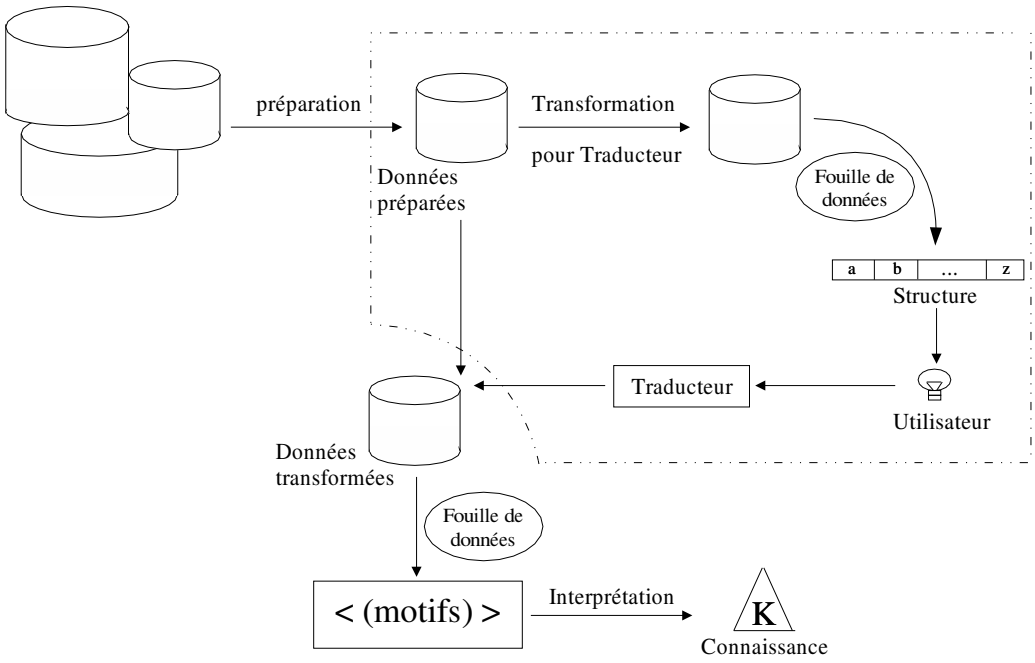

FIG. 12.9 – *Nouveau processus d'ECD*

Reprenons le cas 2, de la section 12.2.2. Si le propriétaire du fichier connaît déjà la transformation qu'il veut appliquer, il n'est pas pour autant en possession du traducteur qui lui permettra de transformer ses données en données du type "`client - date - item`". Pour l'aider dans cette phase, nous proposons de procéder à une analyse du fichier en entrée par un processus de Text Mining destiné à découvrir des éléments communs à toutes les entrées de son fichier. Cette analyse aura pour résultat une structure qui sera alors une base appréciée pour entamer l'écriture d'un parser.

La première phase de cette analyse consiste à associer un numéro de client à chaque ligne du fichier. Dans cette ligne, chaque lettre sera associée à un item et chaque passage d'une lettre à l'autre entraîne l'incrémentation de la date. L'exemple 53 illustre ce concept.

Exemple 53 *une ligne classique d'un fichier access log se présente de cette manière :*
`132.208.12.150 - - [29/Nov/1998:18:02:27 0200] "GET /info/ar.gif HTTP/1.0" 200 1137`
La transformation décrite dans cette section consiste alors à associer à cette ligne un numéro de client, puis à chaque caractère un item. Cela donnera la séquence suivante :
$< (1)(3)(2)(.)(2)(0)(8)(.)(1)(2).(1)(5)(0)(.)(\ )(-)(\ )(-)(\ )([]\ldots >$

Une fois cette transformation effectuée, un processus de fouille de données avec un support de 100% sera capable de proposer une structure pour ce fichier. Les expérimentations que nous avons menées dans le cadre de ces travaux, nous ont permis de trouver la structure suivante pour les fichiers access log :

$< (.)(.)(\ )(-)(\ )(-)(\ )([])(/)(/)(:)(:)(:)(\ )(+)(])(\ )(")(\ )(/)(/)(.)(\ )(/)(.)(")(\ ) >$

Nous pouvons reconnaitre dans cette séquence que les deux premiers items fréquents sont les 2 points qui séparent les éléments d'une adresse IP, mais nous pouvons également noter le fait que cette adresse est toujours suivie de l'enchaînement "espace, tiret, espace, tiret", etc. Une fois mise en parallèle avec le fichier, cette structure peut être d'un grand secours pour mettre au point les règles syntaxiques et grammaticales qui seront la base du parser.

Nous proposons donc deux outils d'aide au prétraitement des données, issus de nos travaux relatifs à la recherche de structures. Le premier consiste à délimiter des champs dans le fichier à analyser. Le second profite des connaissances qu'il a acquise sur ce fichier et sa structure pour proposer un parser dont l'utilisateur n'a pas à se soucier. Si le premier de ces deux outils semble à présent aboutir, le second reste une source d'études encore assez riche. En effet dans le cas 2 de la section 12.2.2, il suffirait que l'utilisateur exprime quels sont les champs qu'il veut garder pour générer un traducteur de façon triviale, cependant en ce qui concerne le cas 3, il y a une telle quantité de traductions possibles que la seule découverte de structure n'est pas suffisante à la génération d'un parser. C'est dans le cas 3 qu'un générateur de règles syntaxiques et grammaticales issues de l'analyse du fichier et de la détection des champs intéressants prendrait l'avantage.

Méthodes de recherche de structures

La première méthode à laquelle on pourrait penser pour rechercher des structures à partir d'un fichier, serait de transformer les données pour en faire une entrée lisible par un processus de fouille de données (comme décrit dans la section 12.2.2) et d'appliquer au résultat obtenu le processus PSP. Nous avons constaté, lors d'une tentative d'application de cette méthode, que PSP ne peut pas répondre en un temps raisonnable à cette requête, en raison d'une trop grande taille des séquences fréquentes. PSP est en effet destiné à des fichiers de très grande taille (il ne dépend pas de la taille du fichier en

entrée, mais seulement de la taille du résultat) mais paye cette performance en cas de séquences fréquentes trop longues. [Jai01] a proposé l'algorithme LCF2 (Long Candidates First) afin de résoudre ce problème et obtenir les séquences fréquentes très rapidement dans le cadre de la recherche de structure.

Nous avons également procédé à un essai qui met en jeu l'algorithme HDM. Cet algorithme propose en effet de rechercher des fréquents en se basant sur des méthodes stochastiques. La recherche de fréquents est alors très efficace quelle que soit leur taille. Nous avons en effet constaté que la version locale (par opposition à distribuée) d'HDM permettait d'obtenir la structure du document en moins d'une minute pour des fichiers de 20 à 200 Mo et des fréquents de longueur 30 à 40 items. Par opposition à ces résultats, signalons que PSP peut demander jusqu'à une heure de traitement pour des fréquents de 40 items (selon le support fixé) et que GSP peut demander jusqu'à plusieurs heures de calculs pour ce même jeu de données.

Articles publiés dans le cadre de cette thèse

Publication dans une revue internationale avec comité de lecture

- *"An Efficient Algorithm for Web Usage Mining".*
  F. Masseglia, P. Poncelet, and R. Cicchetti
  In **Networking and Information Systems**, Vol. 2, N. 5-6, pp. 571-603, December 1999.

Publication invitée dans une revue internationale

- *"Using Data Mining Techniques on Web Access Logs to Dynamically Improve Hypertext Structure".*
  F. Masseglia, P. Poncelet, and M. Teisseire. In **ACM SigWeb Letters**, pp. 13-19, Vol. 8, N. 3, October 1999.

Publications dans des conférences internationales avec comité de lecture

- *"real time web usage mining with a distributed navigation analysis".*
  F. Masseglia and M. Teisseire and P. Poncelet
  In 12th International Workshop on Research Issues on Data Engineering (**RIDE'02**), February 2002, San Jose, USA.
- *"Real Time Web Usage Mining: a Heuristic based Distributed Miner".*
  F. Masseglia and M. Teisseire and P. Poncelet
  In Web Information Systems Ingineering (**WISE'01**), December 2001, Kyoto, Japan.
- *"A General Architecture for Finding Structural Regularities on the Web".*
  P.A. Laur, F. Masseglia, P. Poncelet, and M. Teisseire.
  Proceedings of the 9th International Conference on Artificial Intelligence (**AIMSA'00**), Lecture Notes in Artificial Intelligence, Springer Verlag, Varna, Bulgaria, September 2000.
- *"Schema Mining: Finding Structural Regularity among Semistructured Data".*
  P.A. Laur, F. Masseglia, and P. Poncelet.
  Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (**PKDD'2000**), Poster session, Lecture Notes in Artificial Intelligence, Springer Verlag, Lyon, France, September 2000.
- *"Web Usage Mining: How to Efficiently Manage New Transactions and New Clients".*
  F. Masseglia, P. Poncelet, and M. Teisseire.

Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (**PKDD'2000**), Poster session, Lecture Notes in Artificial Intelligence, Springer Verlag, Lyon, France, September 2000.

- *"WebTool: An Integrated Framework for Data Mining".*
  F. Masseglia, P. Poncelet, and R. Cicchetti.
  Proceedings of the 10th International Conference on Database and Expert Systems Applications(**DEXA'99**), Lecture Notes in Computer Science, Springer Verlag, Vol. 1677, pp. 892-901, Florence, Italy, September 1999.

- *"The PSP approach for mining sequential patterns".*
  F. Masseglia, F. Cathala, and P. Poncelet.
  Proceedings of the 2nd European Conference on Principles of Data Mining and Knowledge Discovery in Databases (**PKDD'98**), Lecture Notes in Artificial Intelligence, Springer Verlag, pp. 176-184, Nantes, France, September 1998.

Publications dans des conférences nationales avec comité de lecture

- *"Web Usage Mining Intersites : Analyse du Comportement des Utilisateurs à Impact Immédiat".*
  F. Masseglia and M. Teisseire and P. Poncelet
  Actes des 17ièmes Journées Bases de Données Avancées (**BDA'01**), Agadir, Maroc, Octobre 2001

- *"Incremental Mining of Sequential Patterns in Large Databases".*
  F. Masseglia, P. Poncelet and M. Teisseire.
  Actes des 16ièmes Journées Bases de Données Avancées (**BDA'00**), Blois, France, Octobre 2000.

- *"Extraction efficace de motifs séquentiels : le prétraitement des données".*
  F. Masseglia, P. Poncelet et M. Teisseire.
  Actes des 15ièmes Journées Bases de Données Avancées (**BDA'99**), pp. 341-360, Bordeaux, France, Octobre 1999.

- *"Analyse du comportement des utilisateurs sur le Web".*
  F. Masseglia, P. Poncelet et R. Cicchetti.
  Actes du 17ième Congrés Informatique des Organisations et Systèmes d'Information et de Décision (**INFORSID'99**), Toulon, France, Juin 1999.

Conférencier invité

- *"Recherche de motifs séquentiels : Algorithmes et Applications".*
  F. Masseglia.
  Journée sur le Data Warehouse et le Data Mining à Marseille, Février 1999.

Rapports de contrat et de recherche

- *"Web Usage Mining: How to Efficiently Manage New transactions and New Customers".*
  F. Masseglia, P. Poncelet, and M. Teisseire.
  Rapport de Recherche LIRMM, 18 pages, Fevrier 2000

– *"Schema Mining : Finding Structural Regularity among Semistructured Data".*
P.A. Laur, F. Masseglia and P. Poncelet.
Rapport de recherche LIRMM, Mai 2000.

– *"Le précalcul appliqué à l'extraction de motifs séquentiels en data mining."*
F. Masseglia. Memoire de DEA.

Bibliographie

[Abi97] S. Abiteboul. Querying Semi-Structured Data. In *Proceedings of International Conference on Database Theory (ICDT'97)*, pages 1–18, Delphi, Greece, January 1997.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. In *Proocedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO'93)*, USA, 1993.

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD Conference*, pages 207–216, Washington DC, USA, May 1993.

[AL97] E. Aarts and JK. Lenstra, editors. *Local Search in Combinatorial Optimization*, chapter 6. Wiley-Interscience Series in Discrete Mathematics and Optimization, 1997.

[ALSS95] R. Agrawal, K. Lin, H.S. Sawhney, and K. Shim. Fast Similarity Search in Presence of Noise, Scaling, and Translation in Time-Series Database. In *Proocedings of the International Conference on Very Large Database*, Switzerland, 1995.

[AMS+95] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. The Quest Data Mining System. In *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining (KDD'95)*, Montreal, Canada, August 1995.

[AP95] R. Agrawal and G. Psaila. Active Data Mining. In *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining*, August 1995.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L Wiener. The Lorel Query Language for Semi-Structured Data. *International Journal on Digital Libraries*, 1(1):68–88, April 1997.

[AS94] R. Agrawal and R. Srikant. Fast Algorithms for Mining Generalized Association Rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, Santiago, Chile, September 1994.

[AS95] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, Tapei, Taiwan, March 1995.

[Bay98] R.J. Bayardo. Efficiently Mining Long Patterns from Databases. In *Proceedings of the 1998 ACM SIGMOD Conference*, pages 85–93, Almaden, USA, 1998.

[BDHS97] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. Adding Structure to Unstructured Data. In *Proceedings of International Conference on Database Theory (ICDT'97)*, pages 336–350, Delphi, Greece, January 1997.

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Belmont, 1984.

[BMUT97] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In *Proceedings of the International Conference on Management of Data (SIGMOD'97)*, pages 255–264, Tucson, Arizona, May 1997.

[CFaM94] d M. Ranganathan C. Faloutsos a and Y. Manolopoulos. Fast Sequence Matching in Time-Series Database. In *Proocedings of the International Conference on Management of Data (SIGMOD'94)*, 1994.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of the IPSJ Conference*, pages 7–18, Tokyo, Japan, October 1994.

[CHNW96] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Update Technique. In *Proceedings of the 12th International Conference on DataEngineering (ICDE'96)*, New-Orleans, Louisiana, March 1996.

[CKL97] D.W. Cheung, B. Kao, and J. Lee. Discovering User Access Patterns on the World-Wide Web. In *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'97)*, February 1997.

[CLK97] D.W. Cheung, S.D. Lee, and B. Kao. A General Incremental Technique for Maintening Discovered Association Rules. In *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications (DASFA'97)*, Melbourne, Australia, April 1997.

[CLR94] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusets, 1994.

[CMS97] R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: Information and Pattern Discovery on the World Wide Web. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, November 1997.

[Com96] Comm. *Data Mining*. Communications of the ACM, 1996.

[Con98] World Wide Web Consortium. httpd-log files. In *http://lists.w3.org/Archives*, 1998.

[Coo00] R.W. Cooley. Web Usage Mining: Discovery and Application of Interesting Patterns from Web Data. In *PHD Thesis*, University of Minnesota, 2000.

[CP97] F. Cathala and P. Poncelet. Preserving Behaviour : Why and How. In *Proceedings of the 9th international Conference on Advanced Information Systems Engineering (CAISE'97)*, pages 333–346, Barcelona, Spain, 1997.

[CR93] A. Califano and I. Rigoutsos. FLASH: A Fast Look-up Algorithm For String Homology. In *Proc. of the 1st Int'l Conference on Intelligence Systems for Molecular Biology*, pages 353–359, Bethesda, July 1993.

[DM85] T. Dietterich and R. Michalski. Discovering patterns in sequences of events. *Artificial Intelligence*, 25:187–232, 1985.

[DTk98] L. Dehaspe, H. Toivonen, and R.D. king. Fining Frequent Substructures in Chemical Coumpounds. In *Proocedings of the 4th International Conference on Knowldege Discovery and Data Mining*, pages 30–36, New York, 1998.

[Dyr97] C. Dyreson. Using an Incomplete Data Cube as a Summary Data Sieve. *Bulletin of the IEEE Technical Committee on Data Engineering*, pages 19–26, March 1997.

[EKSX96] M. Ester, H.P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise. In *Proocedings of the 2nd International Conference on Knowledge Dicovery and Data Mining (KDD'96)*, pages 226–231, Portland, USA, 1996.

[FFKL98] M. Fernández, D. Florescu, J. Kang, and A. Levy. Catching the Boat with Strudel: Experiences with a Web-Site Management System. *Proceedings of the International Conference on Management of Data (SIGMOD'98) - SIGMOD record*, 27(2):414–425, 1998.

[FPSS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. 1996.

[FPSSU96] U.M. Fayad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI Press, Menlo Park, CA, 1996.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Atractability.* Bell Telephone Laboratories, Menlo Park, CA, 1979.

[GMS97] D. Gunopulos, H. Manila, and S. Saluja. Discovering all Most Specific Sentences by Randomized Algorithms Extended Abstract. Technical report, Academy of Finland, 1997.

[GPW98] G. Gardarin, P. Pucheral, and F. Wu. Bitmap Based Algorithms For Mining Association Rules. In *Actes des journées Bases de Données Avancées (BDA'98)*, Hammamet, Tunisie, October 1998.

[GRS98a] S. Guha, R. Rastori, and K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proocedings of the International Conference on Management of Data (SIGMOD'98)*, pages 73–84, Seattle, USA, 1998.

[GRS98b] S. Guha, R. Rastori, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. In *Proocedings of the 1(th International Conference on Data Engineering (ICDE'98)*, Sidney, Australia, 1998.

[HF95] J. Han and Y. Fu. Discovery of Multiple-Level Association Rules from Large Databases. In *Proceedings of the 21 st International Conference on Very Large Databases (VLDB'95)*, pages 420–431, Zurich, Switzerland, September 1995.

[HFW+96] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaïane. Dmql: A Data Mining Query Language for Relational Databases. In *Proceedings of the SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, 1996.

[HGMC+97] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting Semistructured Information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data. See [Wor97]*, Tucson, Arizona, May 1997.

[HKPT98] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient Dicsovery of Functional and Approximate Dependencies Using Partitions. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, USA, 1998.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of International Conference on Management of Data (SIGMOD'2000)*, Dallas, Texas, 2000.

[HS93] M. Houtsma and A. Swami. Set-Oriented Mining of Association Rules. Technical Report RJ 9567, IBM Almaden Research Center, San Jose, California, October 1993.

[HS95] M. Houtsma and A. Swami. Set-Oriented Mining of Association Rules. In *Proceedings of International Conference on Data Engineering (ICDE'95)*, Taiwan, 1995.

[INS] INSEE. rapport 339-340, mai 2001.

[IPS] IPSOS. 12 avril 2001.

[Jai01] S. Jaillet. Aide à la Préparation des Données dans le Processus d'Extraction de Connaissances. Technical report, LIRMM, France, June 2001.

[JD88] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[KMA+98] C.A. Knoblock, S. Minton, J.L. Ambite, N.Ashish, P.J Modi, I. Musla, A.G. Philpot, and S. Tejada. Modeling Web Sources for Information Integration. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 211–218, Madison,Wisconsin, 1998.

[KMRS92] M. Kantola, H. Mannila, K.J. Raiha, and H. Sirtola. Discovering Functional and Inclusion Dependencies in Relational Database. *Journal of Intelligence Systems*, pages 591–607, 1992.

[LAS97] B. Lent, R. Agrawal, and R. Srikant. Discovering Trends in Text Databases. In *Proceedings of the 3rd Int'l Conference on Knowledge*, Newport Beach, California, August 1997.

[Lau00] P.A. Laur. Schema Mining: vers une approche efficace. Technical Report (in preparation), LIRMM, France, June 2000.

[Lie95] H. Lieberman. Letizia: An Agent that Assists Web Browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995.

[LL98] M.Y Lin and S.Y. Lee. Incremental Update on Sequential Patterns in Large Databases. In *Proceedings of the Tools for Artificial Intelligence Conference (TAI'98)*, pages 24–31, May 1998.

[LMP00] P.A. Laur, F. Masseglia, and P. Poncelet. Schema Mining: Finding Structural Regularity among Semistructured Data. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'2000)*, Lyon, France, September 2000.

[LMPT00] P.A. Laur, F. Masseglia, P. Poncelet, and M. Teisseire. A General Architecture for Finding Structural Regularities on the Web. In *Proceedings of the 9th International Conference on Artificial Intelligence (AIMSA'00)*, Varna, Bulgaria, September 2000.

[LPL00] S. Lopez, J-M. Petit, and L. Lakhal. Discovery of Functional Dependencies and Armstrong Relations. In *Proocedings of the 7th International Conference on Extending Database Technology (EDBT'98)*, Germany, March 2000.

[LSM99] W. Lee, S.Stolfo, and K. Mok. Mining in a Data-Flow Environment: Experience in Network Intrusion Detection. In *Proceeding of the 5th International Conference on Knowledge Discovery and Data Mining (KDD'99)*, San Diego, CA, August 1999.

[LYS97] M. Latourrette, L. Yriarte, and J. Sallantin. Structuration d'un jeu de données à l'aide de techniques d'apprentissage : Application au test de robots mobiles. In *JICAA'97: Journées Ingénierie des Connaissances et Apprentissage Automatique*, pages 581–592, Roscoff, France, May 1997.

[MA99] B. Mortazavi-Asl. *Discovering and Mining User Web-Page Traversal Patterns*. PhD thesis, 1999.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. LORE: a Database Management System for Semi-Structured Data. *SIGMOD Record*, 26(3), September 1997.

[Mas98] F. Masseglia. Le pré-calcul appliqué à l'extraction de sequential patterns en data mining. Technical report, LIRMM, France, June 1998.

[MCP98] F. Masseglia, F. Cathala, and P. Poncelet. The PSP Approach for Mining Sequential Patterns. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98), LNAI, Vol. 1510*, pages 176–184, Nantes, France, September 1998.

[MG98] L. Moreau and N. Gray. A Community of Agents Maintening Link Integrity in the World-Wide Web. In *Proceedings of the 3rd International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'98)*, pages 221–233, London, UK, March 1998.

[MJHS96] B. Mobasher, N. Jain, E. Han, and J. Srivastava. Web Mining: Pattern Discovery from World Wide Web Transactions. Technical Report TR-96-050, Department of Computer Science, University of Minnesota, 1996.

[MPC99a] F. Masseglia, P. Poncelet, and R. Cicchetti. An Efficient Algorithm for Web Usage Mining. *Networking and Information Systems Journal*, 2(5-6), December 1999.

[MPC99b] F. Masseglia, P. Poncelet, and R. Cicchetti. Analyse du comportement des utilisateurs sur le Web. In *Actes du 17ième Congrès Informatique des Organisations et Systèmes d'Information et de Décision (INFORSID'99)*, Toulon, France, Juin 1999.

[MPC99c] F. Masseglia, P. Poncelet, and R. Cicchetti. WebTool: An Integrated Framework for Data Mining. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications (DEXA'99)*, Florence, Italy, September 1999.

[MPC00] F. Masseglia, P. Poncelet, and R. Cicchetti. An Efficient Algorithm for Web Usage Mining. *Networking and Information Systems Journal*, April 2000.

[MPT99a] F. Masseglia, P. Poncelet, and M. Teisseire. Extraction efficace de motifs séquentiels : le prétraitement des données. In *Actes des Journées Bases de Données Avancées (BDA'99)*, Bordeaux, France, Octobre 1999.

[MPT99b] F. Masseglia, P. Poncelet, and M. Teisseire. Incremental Mining of Sequential Patterns in Large Databases. In *Actes des Journées Bases de Données Avancées (BDA'00)*, Blois, France, Octobre 1999.

[MPT99c] F. Masseglia, P. Poncelet, and M. Teisseire. Using Data Mining Techniques on Web Access Logs to Dynamically Improve Hypertext Structure. *ACM SigWeb Letters*, 8:13–19, October 1999.

[MPT00] F. Masseglia, P. Poncelet, and M. Teisseire. Web usage mining: How to efficiently manage new transactions and new clients. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'2000)*, Lyon, France, September 2000.

[MTP01a] F. Masseglia, M. Teisseire, and P. Poncelet. Real Time Web Usage Mining: a Heuristic based Distributed Miner (long). In *Web Information Systems Ingineering (WISE'01)*, Kyoto, Japan, December 2001.

[MTP01b] F. Masseglia, M. Teisseire, and P. Poncelet. Web Usage Mining Intersites : Analyse du Comportement des Utilisateurs à Impact Immédiat. In *Base de Données Avancées (BDA'01)*, Agadir, Maroc, October 2001.

[MTP02] F. Masseglia, M. Teisseire, and P. Poncelet. Real Time Web Usage Mining: a Heuristic based Distributed Miner. In *12th International Workshop on Research Issues on Data Engineering (RIDE'02)*, San Jose, USA, February 2002.

[MTV94] H. Mannila, H. Toivonen, and A. I. Verkano. Improved Methods for Finding Assocation Rules. Technical report, University of Helsinki, Finland, February 1994.

[MTV95] H. Mannila, H. Toivonen, and A. I. Verkano. Discovering Frequent Episodes in Sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery in Databases and Data Mining (KDD'95)*, pages 210–215, Montreal, Canada, August 1995.

[MTV97] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3), September 1997.

[Mue95] A. Mueller. Fast Sequential and Parallel Algorithms for Association Rules Mining: A Comparison. Technical Report CS-TR-3515, Department of Computer Science, University of Maryland-College Park, August 1995.

[NAM98] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. *Proceedings of the International Conference on Management of Data (SIGMOD'98) - SIGMOD record*, 27(2), 1998.

[NC00] N. Novelli and R. Cicchetti. Mining Functional and Embedded Dependencies using Free Sets. In *Actes des journées Bases de Données Avancées (BDA'00)*, Blois, France, 2000.

[NUWC97] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative Objects: Concise Representations of Semistructured, Hierarchical Data. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*, pages 79–90, Birmingham, U.K., April 1997.

[NV96] C. Neuss and J. Vromas. *Applications CGI en Perl pour les Webmasters.* Thomson Publishing, 1996.

[OL96] I. Osman and G. Laporte. Metaheuristics: A Bibliography. *Annals of Operations Research*, pages 513–623, 1996.

[PBTL98] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient Mining of Association Rules Using Closed Itemset Lattices. In *Actes des journées Bases de Données Avancées (BDA'98)*, Hammamet, Tunisie, October 1998.

[PBTL99] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules. In *Proceedings of International Conference on Database Theory (ICDT'99)*, pages 398–416, Israel, 1999.

[PHMA+01] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and MC. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In *Proceedings of 17th International Conference on Data Engineering (ICDE'01)*, 2001.

[PMB96] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill and Webert: Indentifying Interesting Web Sites. In *Proceedings of the AAAI Spring Symposium on Machine Learning In Information Access*, Portland, Oregon, 1996.

[PZOD99] S. Parthasarathy, M.J. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and Interactive Sequence Mining. In *Proceedings of the 8th International Conference on Information and Knowledge Management (CIKM'99)*, pages 251–258, Kansas City, MO, USA, November 1999.

[QET97] J. Quinqueton, B. Esfandiari, and R. Terrat. Techniques d'intelligence artificielle distribuée pour la supervision-maintenance des réseaux. *Actes des Séminaires Action Scientifique*, pages 172–193, 1997.

[RF98] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: the teiresias algorithm. *bioinformatic*, 4(1):55–67, 1998.

[RMR96] C. P. Rainsford, M. K. Mohania, and J. F Roddick. Incremental Maintenance Techniques for Discovered Classification Rules. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications*, pages 302–305, Kyoto, Japan, 1996.

[RMR97] C. P. Rainsford, M. K. Mohania, and J. F Roddick. A Temporal Windowing Approach to the Incremental Maintenance of Association Rules. In *Proceedings of the Eighth International Database Workshop, Data Mining, Data Warehousing and Client/Server Databases (IDW'97)*, pages 78–94, Hong Kong. Fong„ 1997.

[SA95] R. Srikant and R. Agrawal. Mining Generalized Association Rules. In *Proceedings of the 21 st International Conference on Very Large Databases (VLDB'95)*, pages 407–419, Zurich, Switzerland, September 1995.

[SA96a] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proceedings of the 1996 ACM SIGMOD Conference*, Montreal, Canada, June 1996.

[SA96b] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, pages 3–17, Avignon, France, September 1996.

[SAM96] J.C. Shafer, R. Agrawal, and M. Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proocedings of the 22th International Conference on Very Large Databases (VLDB'96)*, Bombay, India, September 1996.

[SBKFF] Cyrus Shahabi, Farnoush Banaei-Kashani, Jabed Faruque, and Adil Faisal. Feature matrices: A model for efficient and anonymous mining of web navigations.

[SON95] A. Savasere, E. Omiecinski, and S. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. In *Proceedings of the 21 st International Conference on Very Large Databases (VLDB'95)*, pages 432–444, Zurich, Switzerland, September 1995.

[Sri96] R. Srikant. Fast Algorithms for Mining Association Rules and Sequential Patterns. Technical Report PHD Dissertation, University of Wisconsin - Madison, 1996.

[SS98] N.L. Sarda and N. V. Srinivas. An Adaptive Algorithm for Incremental Mining of Association Rules. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, Indian Institute of Technology Bombay, 1998.

[TBAR97] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti, and Sanjay Ranka. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD '97)*, Newport Beach, California, August 1997.

[Toi96] H. Toivonen. Sampling Large Databases for Association Rules. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB'96)*, September 1996.

[Wan97] K. Wang. Discovering Patterns from Large and Dynamic Sequential Data. *Journal of Intelligent Information System*, pages 8–33, 1997.

[WCM+94] J.T. Wang, G-W Chirn, T. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial Pattern Discovery for Scientific data: Some Preliminary Results. In *Proceedings of the 1994 ACM SIGMOD Conference on Management of Data*, Minneapolis, May 1994.

[WG99] F. Wu and G. Gardarin. Gradual Clustering Algorithms for Metric Spaces. In *Actes des 15ièmes Journées Bases de Données Avancées (BDA'99)*, Bordeaux, France, 1999.

[WK91] S.M. Weiss and C.A. Kulikowski. *Computer System that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufman, 1991.

[WL97a] K. Wang and H.Q. Liu. Mining Nested Association Patterns. In *Proceedings of SIGMOD'97 Workshop on Research Issues on Data Mining and Knowledge Discovery*, May 1997.

[WL97b] K. Wang and H.Q. Liu. Schema Discovery from Semistructured Data. In *Proceedings of International Conference on Knowledge Discovery and Data Mining*, pages 271–274, Newport Beach, August 1997.

[WL98] K. Wang and H.Q. Liu. Discovering Typical Structures of Documents: A Road Map Approach. In *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 146–154, Melbourne, Austrialia, August 1998.

[WL99] K. Wang and H. Liu. Discovering Structural Association of Semistructured Data. *IEEE Transactions on Knowledge and Data Engineering*, 1999.

[WM92] S. Wu and U. Manber. Fast Text Searching Allowing Errors. *Communications of the ACM*, 35(10):83–91, October 1992.

[Wor97] Work97. The Workshop on Management of Semistructured Data. In *www.research.att.com/šuciu/workshop-papers.html*, Tucson, Arizona, May 1997.

[WT96] K. Wang and J. Tan. Incremental Discovery of Sequential Patterns. In *Proceedings of ACM SIGMOD'96 Data Mining Workshop*, pages 95–102, Montréal, Canada, June 1996.

[YDS+96] L. Yriarte, P. Deplanques, J. Sallantin, P. Reitz, R. Zapata, B. Burg, and F. Arlabosse. An architecture for modeling and validation. Application to mobile robotics. In *ECAI'96: 12th European Conference on Artificial Intelligence*, Budapest, Hungary, August 1996.

[Zak98] M. Zaki. Scalable Data Mining for Rules. Technical Report PHD Dissertation, University of Rochester - New York, 1998.

[Zak99] Mohammed J. Zaki. Fast Mining of Sequential Patterns in Very Large Databases. Technical report, The University of Rochester, New York 14627, 1999.

[ZXH98] O. Zaïane, M .Xin, and J. Han. Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs. In *Proceedings on Advances in Digital Libraries Conference (ADL'98)*, Santa Barbara, CA, April 1998.