No d'ordre 04 ISAL 0001 Année 2004

Thèse

Extraction de Motifs Séquentiels Sous Contraintes dans des Données Contenant des Répétitions Consécutives

Présentée devant
L'Institut National des Sciences Appliquées de Lyon

Pour obtenir
Le grade de docteur

École doctorale : Informatique et Information pour la Societé (EDIIS)
Specialité : Informatique - Extraction des Connaissances à partir des Données (ECD)

par
Marion LELEU
(ingénieur)

soutenue le 9 Janvier 2004
devant la commission d'examen

Jean-François BOULICAUT	Directeur	Maître de Conférences HDR à l'INSA de Lyon
Guillaume EUVRARD		Chercheur à Informatique CDC à Paris
Fosca GIANNOTTI		Directeur de Recherche au CNR de Pise (Italie)
François JACQUENET	Rapporteur	Professeur à l'Université Jean Monnet de Saint-Etienne - Eurise
Pascal PONCELET	Rapporteur	Professeur à l'Ecole des Mines d'Alés
Christophe RIGOTTI		Maître de Conférences à l'INSA de Lyon

Cette thèse a été préparée au Laboratoire d'InfoRmatique en Image et Systèmes d'information (FRE 2672 CNRS) de l'INSA de Lyon

Remerciements

Je souhaite exprimer ma sincère gratitude envers Robert Laurini, Professeur à l'INSA de Lyon et co-directeur du laboratoire LIRIS, pour m'avoir accueillie.

Je suis très reconnaissante à François Jacquenet, Professeur à l'Université Jean Monet de Saint-Etienne, et à Pascal Poncelet, Professeur à l'Ecole des Mines d'Alès, d'avoir été rapporteurs de mon mémoire. Je les remercie vivement pour le travail qu'ils ont consacré à son évaluation.

Je désire également remercier Fosca Giannotti, Directrice de recherche au CNR de Pise (Italie), pour m'avoir fait l'honneur d'accepter de participer à mon jury de thèse.

Merci à Jean-François Boulicaut, Maître de conférence HDR à l'INSA de Lyon et responsable de ma thèse au LIRIS, dont les qualités scientifiques et l'honnêteté intellectuelle m'ont beaucoup appris au cours de ce travail de recherche.

Un grand merci à Guillaume Euvrard, Chercheur à Informatique CDC, pour m'avoir suivie et encouragée tout au long de cette thèse et pour avoir su me soutenir durant mes reflexions.

Merci à toute l'équipe de la Direction des Techniques Avancées pour m'avoir accueillie et intégrée dans leur unité, et en particulier à Timo Breideinstein, premier responsable de ma thèse au sein de l'entreprise, ainsi qu'à André Blavier, pour ses conseils avisés d'expert en développement C++.

Je tiens à exprimer un remerciement spécial à Christophe Rigotti, Maître de Conférences à l'INSA de Lyon, sans qui ce travail de thèse ne serait pas le même, pour son amitié et pour m'avoir transmis sa passion pour la recherche.

Je termine en remerciant profondément ma famille et mes amis, pour leur soutien, leurs encouragements, leur amour et leurs richesses qui m'ont permis de mener à bien cette thèse.

Au désert dont l'étoile souffle
Au regard fervant de son voyageur
Le chemin des voix
Dont le parfum d'épices
Éveille les sens
Qui le guideront
Vers les saveurs d'autres oasis
Et la rencontre
Du goût de ses mirages.

Table des matières

1 Introduction **15**

1.1 Extraction de connaissances dans les données 15
1.2 Un processus itératif et interactif 16
1.3 Contribution 18
1.4 Plan du mémoire 22

2 Définitions et Etat de l'Art **23**

2.1 La fouille de données séquentielles 23
2.1.1 Contexte et applications 23
2.1.2 Définitions préliminaires 24
2.2 Recherche de motifs séquentiels sous contraintes 26
2.2.1 La contrainte de fréquence minimum 27
2.2.2 Les contraintes de temps 27
2.2.3 Les contraintes syntaxiques 29
2.2.4 Les autres contraintes 30
2.2.5 Propriété et prise en compte des contraintes 30
2.3 Les différentes familles d'algorithmes 31
2.3.1 Approche type *Apriori* 32
2.3.2 Approche par projections 37
2.3.3 Approche par listes d'occurrences 43
2.4 Synthèse et conclusion 53

3 Algorithmes **57**

3.1 Motivations 58
3.2 L'algorithme *GoSpade* 61
3.2.1 *GoIdLists* : des listes d'occurrences généralisées 61
3.2.2 L'algorithme *GoSpade* 62
3.2.3 Correction de l'algorithme *GoSpade* 67
3.2.4 Résultats expérimentaux 73
3.3 L'algorithme *GoSpec* 78
3.3.1 Les occurrences généralisées contraintes 78
3.3.2 Les opérations de jointure appliquées par *GoSpec* 83

3.3.3 Correction de l'algorithme *GoSpec* 89
3.3.4 Résultats expérimentaux 97
3.4 Bilan 99

4 *GoSpade* et *GoSpec*, en pratique **101**
4.1 Points d'implémentations 101
4.1.1 Gestion de la mémoire 101
4.1.2 Optimisation de la génération contrainte des 2-motifs . . . 105
4.1.3 Optimisation des opérations de *jointure* 105
4.1.4 Gains expérimentaux apportés par ces optimisations 109
4.2 Application au domaine de la finance 112
4.2.1 Signature de situations boursières 115
4.2.2 Classification de situations boursières 121
4.3 Conclusion 122

5 Conclusions et Perspectives **123**

Table des figures

1.1 Le processus d'extraction de connaissances dans des données. . . . 17
1.2 Illustration d'une extraction de motifs séquentiels fréquents. 19
1.3 Les contraintes de temps. 19
1.4 Exemples de répétitions consécutives. 21

2.1 Exemple d'une représentation d'une base de séquences D. 25
2.2 Exemple de candidats générés lors de différentes itérations de l'algorithme 1 (parcours en largeur). 34
2.3 La structure hash-tree de GSP. 35
2.4 Exemple de bases projetées avec PrefixSpan. 40
2.5 Exemple d'un arbre de parcours en profondeur. 41
2.6 Base exemple pour PrefixSpan avec gestion des contraintes. 42
2.7 Exemple de listes d'occurrences *Spade* pour A,B et A $\rightarrow$B avec la base exemple de la figure 2.1. 46
2.8 Illustration des différents cas de génération avec les types de jointures associés. 47
2.9 Exemple d'une jointure temporelle entre les listes $IdList$(A) et $IdList$(B) pour former la liste d'occurrences du motif A $\rightarrow$ B sur la base exemple de la figure 2.1. 49
2.10 Exemple d'une jointure équivalente entre les listes $IdList$(A) et $IdList$(B) pour former la liste d'occurrences du motif AB sur la base exemple de la figure 2.1. 49
2.11 Exemple de listes d'occurrences contraintes pour A, A $\rightarrow$ C et A $\rightarrow$ C $\rightarrow$ B avec la base exemple de la figure 2.1. 51
2.12 Synthèse des algorithmes existants (**ER** = Expression Régulière). . 53
2.13 Origine de la consommation mémoire et des coûts de calculs pour chaque famille d'algorithmes. 54

3.1 Exemple base contenant des répétitions consécutives. 58
3.2 Listes d'occurrences *Spade* vs. Listes d'occurrences compactées. . . 59
3.3 Evolution du temps d'exécution global de *Spade* sur des jeux contenant des répétitions consécutives. 60

3.4 Longueur moyenne des listes d'occurrences de *Spade* vs. longueur moyenne listes d'occurrences généralisées de *GoSpade*. 61
3.5 Algorithme général (*GoSpade*). 63
3.6 Algorithme de fusion. 65
3.7 Algorithme de jointure. 66
3.8 Algorithme de jointure temporelle ($TpJoin$). 67
3.9 Algorithme de jointure temporelle locale ($LocalTpJoin$). 68
3.10 Algorithme de jointure équivalente ($EqJoin$). 69
3.11 Algorithme de jointure équivalente locale ($LocalEqJoin$). 70
3.12 Evolution du nombre total d'occurrences utilisées. 75
3.13 Evolution du nombre total de comparaisons. 76
3.14 Evolution du temps d'exécution total. 77
3.15 Influence des répétitions consécutives sur *Spade* et *GoSpade*. . . . 77
3.16 Base de séquences contenant des répétitions consécutives. 79
3.17 Listes d'occurrences et listes d'occurrences généralisées contraintes pour les motifs A, B, C, A $\rightarrow$ B, A $\rightarrow$ C et A $\rightarrow$ B $\rightarrow$ C de la base de la figure 3.16, avec les contraintes suivantes : $minSup = 2$, $minGap = 2$, $maxGap = 5$ et $maxWin = 10$. 80
3.18 Algorithme de jointure temporelle contrainte ($cTpJoin$). 84
3.19 Algorithme de jointure temporelle locale contrainte ($cLocalTpJoin$). 85
3.20 Illustration des tests d'entrée de l'algorithme 12 ($cLocalTpJoin$). . 86
3.21 Algorithme de jointure temporelle spécifique ($cTpJoinLevel_2$). . . 88
3.22 Algorithme de jointure équivalente contrainte ($cEqJoin$). 89
3.23 Algorithme de jointure équivalente locale contrainte ($cLocalEqJoin$). 90
3.24 Evolution du nombre total d'occurrences utilisées. 98
3.25 Evolution du nombre total de comparaisons effectuées. 98
3.26 Evolution du temps d'exécution. 99
3.27 Influence des répétitions consécutives sur *cSpade* et *GoSpec*. 99

4.1 Algorithme général *GoSpade* optimisé. 102
4.2 Algorithme $cTpJoinLevel_2$ optimisé. 104
4.3 Passage de listes contraintes ($occList$) à des listes non contraintes ($GoIdList$). 109
4.4 Effet des optimisations. 110
4.5 Gain obtenu sur la taille des listes d'occurrences. 110
4.6 Gain obtenu sur le nombre de jointures effectuées. 111
4.7 Gain obtenu sur le temps d'exécution. 113
4.8 Evolution du temps d'exécution lorsque la taille des bases de séquences augmentent ($minSup$= 15%, $maxGap$= 4). 113
4.9 Comportement de *GoSpec* sur des données biologiques. 114
4.10 Comportement de *GoSpec* sur des données financières. 114
4.11 Signature de situations boursières : principe. 116
4.12 cours-MME-Sigma mobile. 117

4.13 Signification des alphabets. 118
4.14 Exemple de pré-traitement de données qualitatives (dépêches de presse). 119
4.15 Description des signatures obtenues. 120
4.16 Principe de classification de situations boursières. 121
4.17 Résultats de classification de situations boursières. 122

Chapitre 1

Introduction

1.1 Extraction de connaissances dans les données

Durant les dernières décennies, les volumes d'informations collectés au sein des entreprises ont considérablement augmenté. Cette augmentation a donné lieu à la création de bases de données de plus en plus nombreuses et de plus en plus volumineuses dans de nombreuses disciplines (biologie, finance, réseaux, Internet, commerce, ...).

Pour faire face à cette multiplication des informations stockées, il a fallu, dans un premier temps, développer des techniques permettant d'exploiter efficacement ces bases de données. Cette efficacité reposait d'une part sur la qualité des informations conservées dans les bases et d'autre part sur la qualité et la facilité d'utilisation des moyens fournis aux utilisateurs pour accéder à ces informations. Pour cela, un certain nombre de méthodes de conception de bases de données ont été développées afin de s'assurer que les informations conservées seraient utiles et pertinentes pour les utilisateurs. Dans un même temps, les outils d'interrogation ont été améliorés pour permettre à ces utilisateurs d'accéder plus facilement et plus efficacement à ces informations (e.g., langages graphiques et langages de requêtes dédiés à certains domaines).

Par la suite, les besoins d'analyse pouvaient porter sur des informations provenant de plusieurs bases de données (e.g., analyse des ventes par région à partir de bases provenant de plusieurs magasins). C'est le regroupement de plusieurs bases de données en entrepôts de données qui a permis de fournir aux utilisateurs une source uniformisée de données d'origines mixtes. Dans ces entrepôts, les outils d'interrogations classiques ne sont plus suffisamment efficaces pour répondre aux différents niveaux de granularité souhaités par les utilisateurs (e.g., analyse des moyennes des ventes par ville, par département et par région) et demandent des temps de calculs trop longs sur ces volumes de données. De nouvelles structures

spécifiques à ces besoins d'analyse ont alors été développés, ainsi que des outils dédiés au traitement de ces données à différents niveaux de granularité, comme par exemple les systèmes OLAP (On Line Analytical Processing).

Toutes ces techniques et outils d'analyse ont prouvé leur efficacité mais les besoins des utilisateurs se sont orientés vers un désir d'avoir une vision plus synthétique des informations et des connaissances contenues dans leurs données, telles que les 'lois', les tendances ou encore les phénomènes caractéristiques qu'elles contiennent. Le problème a ainsi évolué vers des besoins dont l'objectif n'est plus de répondre à une demande précise (e.g., *Quel est le nombre de clients ayant acheté un produit donné ?*), mais à des questions plus générales (e.g., *quelles sont les particularités des clients ayant acheté un produit donné ?*).

C'est cette demande qui a donné naissance à une nouvelle discipline informatique, au milieu des années 1990, appelée *fouille de données* (ou plus communément *Data Mining* en anglais [FPSSU96, HK00, Gar99, Gus97, FPSS96a, IM96]). Celle-ci est apparue comme un domaine fédérateur ([FPSS96b]) permettant de regrouper des équipes de différentes disciplines (e.g., statistiques, analyses de données, bases de données, apprentissage) qui auparavant travaillaient indépendamment sur ces problèmes. La fouille de donnée s'inscrit dans un processus plus complet, appelé *extraction de connaissances dans les données* (ou *knowledge discovery in databases*, en anglais), qui comprend plusieurs étapes telles que la collecte des données, la recherche des 'lois' et le traitement des résultats obtenus. Ces différentes étapes sont décrites dans la section suivante.

1.2 Un processus itératif et interactif

Le processus d'extraction de connaissances dans les données a été défini dans [FPSM91] comme étant *le processus non trivial d'extraction d'informations valides, nouvelles, potentiellement utiles, et compréhensibles à partir de données.*

Ce processus comprend cinq étapes comme illustré sur la figure 1.1 : la consolidation, la sélection et le pré-traitement des données, la fouille de données et l'interprétation des résultats. Ce processus est itératif dans la mesure où les résultats d'une étape peuvent remettre en cause les traitements effectués lors des étapes précédentes. De plus, il est interactif puisqu'il ne peut fournir de bons résultats sans intervention des utilisateurs.

- *Consolidation :* Cette première étape permet de regrouper et de mettre en forme des données réelles d'origines diverses (achat, comptabilité, évolution des différents cours boursiers, dépêches de presse, résultats d'analyses financières,...) afin de les intégrer dans ce que l'on appelle un entrepôt de données qui servira de source de référence au processus. Les données contenues dans ces entrepôts peuvent être de natures très diverses. Par exemple, dans le domaine de la finance, celles-ci peuvent être quantitatives comme

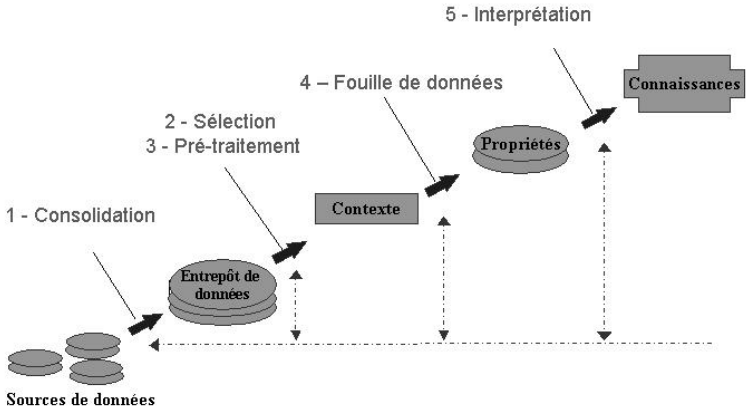


FIG. 1.1 – Le processus d'extraction de connaissances dans des données.

l'évolution d'un indice boursier ou qualitatives (discrètes), comme les dépêches financières. Les liens présentés par ces données peuvent aussi être temporels (différentes dates d'achats dans un magasin) ou spatiaux (phénomènes situés à différentes localisations géographiques).

– *Sélection :* Cette deuxième étape permet de choisir dans l'entrepôt les données sur lesquelles on va travailler. Par exemple, cette étape permet de sélectionner les indices boursiers qui nous intéressent ainsi que les périodes de temps sur lesquelles on souhaite travailler.

– *Pré-traitement :* Cette étape, aussi appelée étape de *nettoyage des données* correspond à une part importante du processus puisqu'elle a pour objectif de s'assurer de la qualité des données qui seront traitées. Par exemple, en corrigeant les données lorsque celles-ci contiennent des valeurs aberrantes, ou en faisant le choix de remplacer ou d'ignorer les données manquantes. L'autre objectif important de cette étape est de représenter les données dans un format adéquat aux besoins spécifiques des algorithmes qui seront appliqués par la suite. Ce deuxième point dépend de la nature de données et de ce que l'on souhaite en faire. Par exemple, dans notre contexte financier, le choix a été fait de représenter les données sous forme de séquences d'événements financiers constituées d'une part par les évolutions de certains indices boursiers (données quantitatives discrétisées) et d'autre part, par les dépêches de presse fournies par des agences spécialisées (données qualitatives représentées par des ensembles de mots-clés).
– *fouille de données (*data mining*) :* Cette étape représente le coeur du processus d'extraction de connaissances dans des données. C'est en effet cette

étape qui va procéder à la recherche des 'lois', à proprement dites, contenues dans les données. Les algorithmes appliqués dépendent des objectifs souhaités. Par exemple, à partir de bases de séquences d'événements financiers, on peut être intéressé par trouver des régularités dans les situations de marché. Dans ce cas, un algorithme d'extraction de motifs séquentiels sera appliqué et les motifs caractéristiques obtenus pourront être utilisés pour expliquer un ensemble de situations ou encore servir de base pour classifier des situations de marché futures.
C'est aussi cette étape qui représente le plus de difficultés du point de vue complexité algorithmique. En effet, les algorithmes à développer doivent être capables de traiter en des temps raisonnables de grands volumes de données et parcourir de façon efficace de très grands espaces de recherche. L'utilisation de contraintes de façon active lors de cette étape permet de réduire les espaces de recherche et donc les temps d'exécution, tout en permettant à l'utilisateur d'effectuer des recherches plus ciblées (e.g., contraintes de contenu et de durée des motifs à extraire).

– *Interprétation :* Cette étape correspond à l'analyse des résultats obtenus par l'étape de fouille de données (étape 4). Si les utilisateurs estiment que les résultats sont trop généraux et qu'ils ne leur apportent pas d'informations nouvelles ou pertinentes, le choix peut être fait de recommencer le processus à partir de l'étape de sélection (étape 2) afin de concentrer les recherches sur une partie de données plus précise. Par exemple, les séquences d'événements pourront être construites en ne considérant plus un indice boursier trop général, mais un indice plus spécifique à un secteur d'activité.
Le processus peut aussi être relancé à partir de l'étape de pré-traitement des données (étape 3) en reprenant par exemple les seuils appliqués pour discrétiser les cours boursiers et en les ajustant afin d'obtenir des événements discrets mieux répartis, ou encore en regroupant ou en supprimant certains mots-clés utilisés pour représenter les dépêches financières.

1.3 Contribution

Notre contribution concerne le domaine de la recherche de motifs séquentiels fréquents dans une base de séquences. Celui-ci a pour objectif d'identifier toutes les sous-séquences, c'est-à-dire toutes les successions d'événements, qui apparaissent dans une certaine proportion des séquences de la base. Par exemple, considérons la base représentée sur la figure 1.2. Une extraction effectuée à un seuil de 50% va permettre de retrouver tous les motifs séquentiels qui apparaissent dans au moins 2 séquences sur les 4 séquences qui composent cette base. Dans un tel contexte, les motifs A $\rightarrow$ B et BC $\rightarrow$ A seront considérés comme fréquents puisqu'ils apparaissent respectivement dans toutes les séquences et dans 2 séquences sur 4 ($2^{ème}$

et $4^{ème}$ séquence). Par contre, le motif A $\rightarrow$ B $\rightarrow$ C ne sera pas retenu comme motif résultat puisqu'il n'apparaît que dans la première séquence ce qui n'est pas suffisant pour être considéré comme fréquent dans cet exemple.

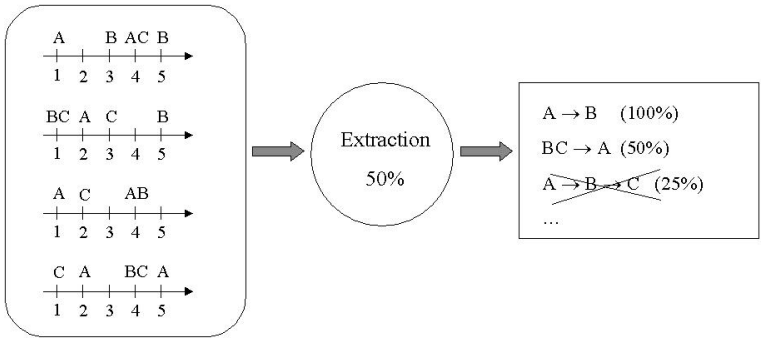


FIG. 1.2 – Illustration d'une extraction de motifs séquentiels fréquents.

Lors d'une recherche de motifs séquentiels fréquents, il est possible d'introduire d'autres contraintes en plus de la contrainte de fréquence. Dans notre travail de thèse nous avons travaillé avec des contraintes temporelles (figure 1.3) comme le gap minimum et le gap maximum, qui permettent de limiter le temps qui s'écoule entre l'apparition de deux événements successifs d'un motif, et la contrainte de fenêtre de temps maximum qui permet de limiter la durée totale d'un motif, c'est-à-dire le temps qui s'écoule entre l'apparition du premier et du dernier événement d'un motif.

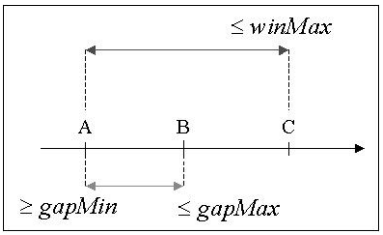


FIG. 1.3 – Les contraintes de temps.

De nombreux efforts ont été fournis ces dernières années pour tenter de traiter des données séquentielles le plus efficacement possible.

Nous avons étudié différentes techniques d'extraction de motifs séquentiels avec contraintes et cette étude nous a permis de faire apparaître trois familles d'algorithmes. Ceux-ci peuvent être distingués par les approches spécifiques développées pour parcourir l'espace des motifs candidats d'une part, et pour évaluer la fréquence

des candidats d'autre part.

La première famille contient les premiers algorithmes développés dans le domaine (e.g., GSP [SA96]). Ceux-ci s'appuient sur une propriété de la fréquence, appelé anti-monotonicité, et qui permet de ne pas chercher à compléter une succession d'événements si celle-ci n'apparaît pas suffisamment souvent dans la base de séquences. Ces algorithmes procèdent à une génération complète des motifs candidats d'un même niveau (i.e., composés d'un même nombre d'événements) avant de compter leur support en effectuant une passe complète sur la base de séquences. Ces deux caractéristiques (génération par niveau et utilisation de l'anti-monotonie de la fréquence) ont été initialement exploités par l'algorithme APriori ([AMS+96, AS94a]) pour la recherche de motifs fréquents dans des données sans composante temporelle. En référence à cet algorithme antérieur bien connu, nous appellerons cette première famille d'approches *famille APriori*.
Les deux autres familles regroupent des algorithmes proposés pour remédier aux limites de la première famille. Ainsi, les algorithmes de la deuxième famille cherchent principalement à réduire les coûts liés à l'étape de génération des motifs candidats, en ne générant que les candidats qui apparaissent au moins une fois dans la base. Ces algorithmes utilisent une approche dite *par projection* (e.g., PrefixSpan [PHMAP01]), et travaillent sur des bases de plus en plus petites, pour lesquelles le coût de comptage est plus restreint.
Enfin, la dernière famille se compose d'algorithmes dits basés sur une approche par *Listes d'occurrences*. Parmi ces algorithmes on peut citer l'algorithme *Spade* [Zak01, Zak98] et sa version contrainte *cSpade* [Zak00], les premiers à avoir utilisé cette approche dans le cas d'extractions effectuées sur des bases de séquences. Le principe est de conserver les apparitions des motifs en mémoire, ce qui permet de réduire les coûts liés aux accès disque lors de l'étape de recherche et de comptage des motifs candidats.

Tous ces algorithmes ont montrés leur efficacité dans différentes situations, en particulier ceux de la troisième famille. Cependant, leurs performances peuvent être grandement améliorées lorsque les données comportent des répétitions consécutives. Par exemple, un client peut acheter plusieurs fois le même article lors d'achats successifs. Ces situations peuvent se produire dans de nombreux contextes applicatifs, et notamment lorsque les séquences sont en partie construites à partir de données quantitatives, telles des séries temporelles discrétisées (cf. cadre du haut, figure 1.4), ou encore dans le cas typique de vocabulaire d'événements réduits (e.g., les séquences d'ADN, cf. cadre du bas, figure 1.4), ce qui augmente les probabilités d'apparition d'un même symbole de manière consécutive.

Dans cette thèse, nous tentons d'apporter des solutions efficaces au problème de l'extraction, contrainte ou non, de motifs séquentiels, dans le cas de données contenant des répétitions consécutives. La solution apportée repose sur une généralisation des listes d'occurrences qui permet de condenser les informations contenues dans

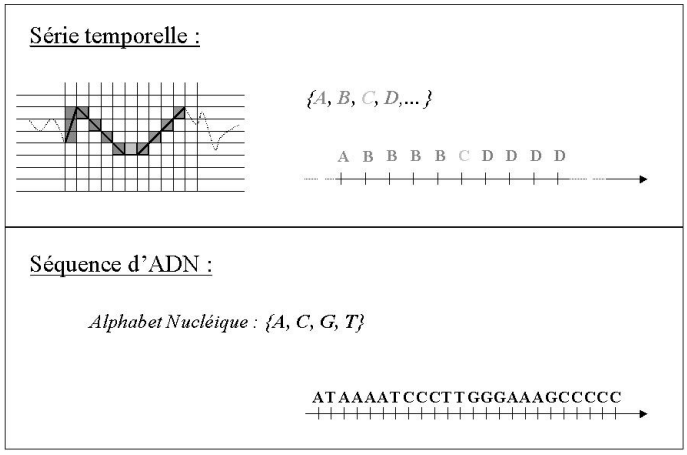


FIG. 1.4 – Exemples de répétitions consécutives.

ces listes, sans perte pour les extractions.

Cette nouvelle représentation a donné lieu aux développements d'extracteurs de motifs séquentiels, *GoSpade* (traitement de la seule contrainte de fréquence minimum) et *GoSpec* (introduction de contraintes temporelles), particulièrement bien adaptés à la présence de répétitions consécutives dans les données. Les algorithmes correspondants ont respectivement fait l'objet d'une démonstration de justesse et de complétude afin d'assurer la correction des résultats qu'ils retournent. Nous avons procédé à des expérimentations sur des jeux de données réelles et synthétiques qui montrent une nette amélioration des performances en présence de répétitions consécutives (cf. figure 3.15 page 77). Les gains obtenus, en terme d'espace mémoire et de temps d'exécution, permettent de travailler sur des volumes de données plus importants et à des seuils de fréquence plus faibles.

Nous avons aussi travaillé sur des techniques d'implémentation efficaces de ces algorithmes, qui nous ont permis de réduire plus encore l'utilisation de l'espace mémoire et d'obtenir des temps d'exécution significativement plus intéressants.

Enfin, dans le cadre industriel de cette thèse (thèse CIFRE à la Caisse des Dépôts et Consignations), nous avons procédé à une application réelle dans le domaine des marchés financiers où l'évolution des marchés financiers est représentée par des séquences d'événements. L'objectif était de construire une représentation synthétique de différentes tendances boursières sous la forme de motifs séquentiels caractéristiques. Nous avons pu montrer que les motifs fréquents représentant une tendance contiennent une information qui est bien spécifique de la tendance correspondante.

1.4 Plan du mémoire

Le chapitre 2 développe un état de l'art des différentes techniques d'extraction de motifs séquentiels sous contraintes. Le chapitre 3 présente deux algorithmes développés au cours de ce travail de thèse permettant d'extraire ce type de motifs, *GoSpade* et *GoSpec* particulièrement bien adaptés à la présence de répétitions consécutives dans les données. Le chapitre 4 introduit des techniques d'implémentation efficaces de ces algorithmes et décrit une application, dans le domaine des marchés financiers, dont l'objectif est d'obtenir une représentation synthétique de différentes tendances boursières sous forme de motifs séquentiels caractéristiques. Enfin, nous concluons au chapitre 5 et présentons certaines suites possibles à ce travail.

Chapitre 2

Définitions et Etat de l'Art

2.1 La fouille de données séquentielles

2.1.1 Contexte et applications

Les données séquentielles sont des données sur lesquelles il existe une notion d'ordre qui peut être spatiale ou temporelle. Par exemple, les séquences d'ADN, les séquences de protéines, ou encore les chaînes de caractères ([BT01, PS00, AHKV97, LAS97, GNPS03, PCGS02]), sont des données composées d'éléments (resp. nucléotides, acides aminés et caractères alphabétiques) ordonnés de manière spatiale. Les données temporelles, quant à elles, peuvent être des successions d'achats de clients ou de pages visitées sur un site Web ([KB00, Coo00, MPC99, MA01, CMS97, GRSS99]), des historiques d'erreurs ou de connexions sur un réseau ([Klm99, HKM+96, MTV97, ZLO98]), ou encore des évolutions de cours boursiers ou de mesures géophysiques ([Pov99, BSWJL98, BC96, HDY99, LB02a, LB02b]).

L'analyse de ces données, ayant pour seul point commun leur caractère ordonné, se traduit souvent par la recherche de motifs pouvant prendre des formes bien différentes. D'une façon générale, ces motifs peuvent correspondre à des successions, contiguës ou non, d'éléments ou de groupes d'éléments, dans le temps ou dans l'espace. Par exemple, le traitement de chaînes de caractères peut aboutir à la recherche de sous-chaînes fréquentes, c'est-à-dire à une succession contiguë de caractères alphabétiques alors que l'analyse des achats effectués par certains clients dans un magasin se traduira par la recherche de successions d'ensemble de produits pour lesquels la proximité imposée ne sera pas spatiale, mais temporelle. Il est aussi possible de s'intéresser à d'autres types de motifs, comme les motifs séquentiels logiques ([MJ02, LDR02]).

Ainsi, le traitement de données séquentielles peut aboutir à une multitude de choix, aussi bien en ce qui concerne la nature des données traitées que les motifs

recherchés. Pour toutes ces raisons il est nécessaire de fixer un cadre de travail bien défini.

Dans cet état de l'art nous ne considérerons pas des approches très spécifiques portant sur des motifs dédiés et utilisant des algorithmes Ad hoc, tel que l'on peut en trouver par exemple dans le domaine de l'analyse de chaînes. Nous considérerons le cadre général de la recherche de motifs séquentiels, apparu en 1995 avec les algorithmes APrioriSome et APrioriAll ([AS95]). A l'origine, ces algorithmes ont été proposés dans le cadre d'une étude de marché portant sur l'analyse des séquences d'achats effectuées par des clients dans un magasin. L'objectif était d'identifier toutes les successions d'achats représentatives des habitudes de consommation des clients d'un magasin.

Les problèmes rencontrés concernaient principalement la taille des bases de séquences à traiter, ainsi que la complexité des calculs qui en découlait. Depuis, de nombreux algorithmes ont été proposés (e.g., [SA96, MCP98, Mas02, HHMAC+00, PHMAP01, Zak01, Zak98, Zak00, AFGY02, GRK99, ALB03, PHW02]) permettant de gérer d'importantes bases de séquences en terme de volume, et ont été utilisés dans de nombreuses autres applications.

L'objectif de l'extraction de motifs séquentiels est de rechercher toutes les successions d'événements qui satisfont une combinaison de critères (appelés *contraintes*) dans un ensemble de séquences.

2.1.2 Définitions préliminaires

Dans cette section, nous rappelons un ensemble de définitions qui permettent de fixer le cadre général de la recherche de motifs séquentiels. Celles-ci concernent d'une part la structure des données traitées (séquences d'événements et bases de séquences), et d'autre part le type de motifs recherché (motif, sous-séquence, occurrence d'un motif). Ces définitions seront ensuite utilisées lors de notre présentation des différents travaux.

Définition 1 ***(Séquence d'événements)*** Soit $I = \{i_1, i_2, \ldots, i_m\}$, un ensemble de m symboles distincts appelés *items* et muni d'un ordre total. Un *événement* (ou *itemset*) de taille l est un ensemble non vide constitué par l items provenant de I. L'ensemble des événements est muni d'un ordre lexicographique défini à partir de l'ordre total des items. Une *séquence d'événements* (ou *séquence*) α de *longueur* L est une liste ordonnée composée de L événements $\alpha_1, \ldots, \alpha_L$, et représentée de la façon suivante : $\alpha_1 \rightarrow \alpha_2 \rightarrow ... \rightarrow \alpha_L$. Dans une séquence, chaque événement possède un identifiant, noté *eid*, qui correspond à sa position dans la liste ordonnée.

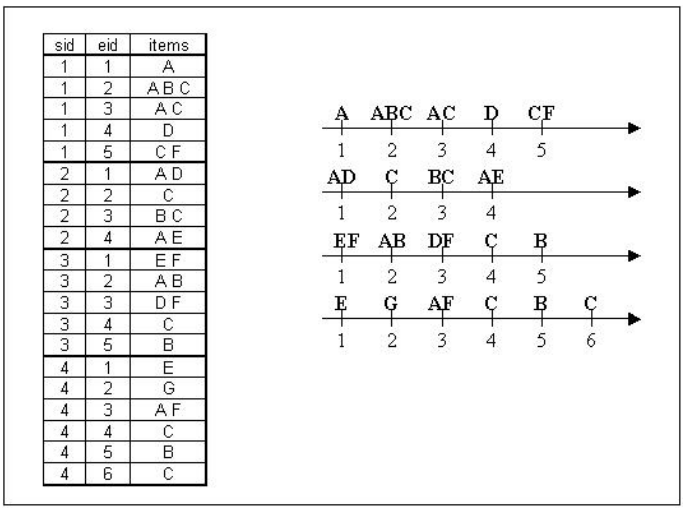
Selon les applications, les événements ne sont pas identifiés par leur valeur de position, mais par leur date d'apparition. Ainsi, dans le domaine de séquences d'achats, les événements (aussi appelés *transactions*), représentent l'ensemble des items achetés par un client à une date donnée. Une séquence ainsi constituée

représente alors les successions d'achats effectués par un client au cours d'une période de temps.

Dans nos exemples, nous utiliserons des lettres majuscules pour dénoter les items, et nous utiliserons l'ordre alphabétique comme ordre total. Ainsi, un événement tel que {E, A, C} sera noté ACE (lorsque cela n'entraîne pas d'ambiguïté, nous omettrons les accolades).

Définition 2 ***(Base de Séquences)*** *Une base de séquences est un ensemble de paires (sid,s) où s représente une séquence et sid correspond à son identifiant.*

Dans une base de séquences, chaque séquence est représentée par un ensemble de tuples ⟨sid,eid,items⟩ correspondants aux événements qui la composent. Les champs *sid* et *eid* correspondent respectivement aux identifiants de la séquence et de l'événement concerné, et *items* représente la liste des items qui composent cet événement. Ainsi, une base de séquences peut être représentée comme illustré dans le tableau gauche de la figure 2.1 (les séquences correspondantes sont données dans la partie droite de cette même figure).



sid	eid	items
1	1	A
1	2	A B C
1	3	A C
1	4	D
1	5	C F
2	1	A D
2	2	C
2	3	B C
2	4	A E
3	1	E F
3	2	A B
3	3	D F
3	4	C
3	5	B
4	1	E
4	2	G
4	3	A F
4	4	C
4	5	B
4	6	C

FIG. 2.1 – Exemple d'une représentation d'une base de séquences D.

Définition 3 ***(Motif)*** Un *motif* est une liste ordonnée d'événements non datés, représenté de la façon suivante : $\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$.

Définition 4 ***(Sous-séquence)*** Une séquence, ou un motif, de la forme $\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$ est appelée une *sous-séquence* d'une séquence $\beta_1 \rightarrow \beta_2 \rightarrow \ldots \rightarrow \beta_m$ s'il existe des entiers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ tels que $\alpha_1 \subseteq \beta_{i_1}$, $\alpha_2 \subseteq \beta_{i_2}$, ..., $\alpha_n \subseteq \beta_{i_n}$.

Par exemple, les motif $B \rightarrow E \rightarrow B$ et $AD \rightarrow B$ sont deux sous-séquences, de la séquence $ABD \rightarrow AE \rightarrow B$. En effet, pour $B \rightarrow E \rightarrow B$, on a $B \subseteq ABD$, $E \subseteq AE$ et $B \subseteq B$, et le motif $AD \rightarrow B$ est tel que $AD \subseteq ABD$ et $B \subseteq B$. Les sous-séquences contiguës correspondent typiquement aux sous-chaînes.

Définition 5 ***(Occurrence d'un motif)*** Si un motif z est sous-séquence d'une séquence s, on dit que z apparaît dans s. L'apparition d'un motif dans une séquence particulière est appelée *occurrence d'un motif* et correspond à une liste ordonnée d'événements accompagnés de leur identifiant dans cette séquence. Une occurrence d'un motif se représente de la façon suivante : $\alpha_1(eid(\alpha_1)) \rightarrow \alpha_2(eid(\alpha_2)) \rightarrow \ldots \rightarrow \alpha_n(eid(\alpha_n))$.

Par exemple, l'occurrence A(1) $\rightarrow$ C(2) $\rightarrow$ D(4) représente une occurrence du motif A $\rightarrow$ C $\rightarrow$ D dans la première séquence de la base exemple de la figure 2.1.

Définition 6 ***(k-motif, préfixe et suffixe d'un motif)*** Un motif $\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$ est un *k-motif* si $\sum_{i=1\ldots n} |\alpha_i| = k$ (i.e., il est composé de k items). Le *suffixe* d'un motif est le plus grand item (pour l'ordre total des items) contenu dans le dernier événement du motif. Le *préfixe* d'un motif est le motif privé de son préfixe.

Par exemple, le motif A $\rightarrow$ C $\rightarrow$ BC est un *4-motif*, dont le *préfixe* correspond au sous-motif A $\rightarrow$ C $\rightarrow$ B, de taille 3, et son *suffixe* correspond à l'item C.

Définition 7 ***(longueur et largeur d'un motif)*** La *longueur* d'un motif $\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$ est n (i.e., le nombre d'événements qu'il contient) et sa *largeur* est $max_{i=1\ldots n}|\alpha_i|$.

Par exemple, le motif C $\rightarrow$ ADE, est un motif de longueur 2 et de largeur 3.

2.2 Recherche de motifs séquentiels sous contraintes

Initialement, la recherche de motifs séquentiels se limitait à l'utilisation d'un seul critère de sélection. Celui-ci correspondait à un seuil de fréquence minimum qui permettait de rechercher toutes les successions d'événements qui se retrouvaient suffisamment souvent dans un ensemble de séquences. Ce critère a été rapidement complété par de nombreux autres critères de sélections, apparus suite à l'expression de nouveaux besoins par les utilisateurs de données. En effet, dans de nombreux domaines d'application, comme les études de marchés, la biologie ou encore la finance, les motifs séquentiels fréquents n'étaient plus suffisants pour caractériser les données. Par exemple, dans le domaine de la finance, l'apparition d'un événement rare pouvait expliquer à lui seul la tendance des données. L'introduction de nouvelles contraintes, intervenant souvent en complément de la contrainte

de fréquence minimum, a donnée lieu à l'adaptation ou à la création de nouveaux algorithmes dont les résultats correspondaient plus aux besoins spécifiques des applications.

Les efforts effectués ont permis non seulement d'obtenir des résultats de meilleurs qualité car les ensembles de motifs obtenus répondaient plus aux attentes des utilisateurs, mais aussi d'améliorer considérablement les processus d'extraction puisque l'espace de recherche pouvait se trouver réduit par l'introduction de certaines de ces contraintes.

Dans cette section, nous allons présenter de façon non exhaustive les différents types de contraintes, en commençant par la contrainte de fréquence minimum, qui peuvent être prises en compte lors d'un processus d'extraction de motifs séquentiels sous contraintes.

2.2.1 La contrainte de fréquence minimum

Dans la littérature (e.g., [SA96]), la notion de motif fréquent se définie comme suit :

Définition 8 ***(Motif Fréquent)*** Soient D une base de séquences et $minSup$ un entier positif appelé *seuil de fréquence absolu*. Soit z un motif et $support(z)$ le nombre de séquences de D dans lequel il apparaît. Le motif z vérifie la contrainte de fréquence minimum dans une base de données D si z est une sous-séquence d'au moins $minSup$ séquences de D, i.e. si $support(z) \geq minSup$. Dans ce mémoire, nous utiliserons aussi le terme *seuil de fréquence relatif*, exprimant le pourcentage de la fréquence absolue en fonction du nombre total de séquences de D.

Exemple 1 *En reprenant la base exemple de la figure 2.1. Supposons que le seuil de fréquence absolu soit de 3, ce qui correspond à un seuil de fréquence relatif de 75% puisque le nombre total de séquences dans la base est 4 (3/4). Ainsi, un motif doit apparaître dans au moins 3 séquences de la base pour être considéré comme fréquent. Par exemple, le motif* $A \rightarrow C \rightarrow B$ *est fréquent puisqu'il apparaît dans les trois dernières séquences de la bases, alors que le motif* $A\ B \rightarrow C$ *n'est pas fréquent puisqu'il n'apparaît que dans deux séquences au total (*$sid = 1$ *et* $sid = 3$*).*

La contrainte de fréquence minimum représente le premier type de contraintes exploité lors d'extractions de motifs séquentiels, et malgré l'apparition d'autres critères de sélection, celle-ci compte parmi les contraintes les plus utilisées.

2.2.2 Les contraintes de temps

Ces contraintes permettent d'agir sur les particularités temporelles des motifs extraits. Il est ainsi possible de ne s'intéresser qu'aux motifs dont la durée ne

dépasse pas un certain intervalle de temps (*fenêtre de temps maximum*), ou encore dont le laps de temps qui s'écoule entre deux événements soit supérieur (resp. inférieur) à une certaine valeur (*gap minimum* (resp. *gap maximum*)).

Ces trois contraintes ont la particularité d'agir au niveau de la validité des occurrences d'un motif et non pas au niveau du motif lui même.

Définition 9 Soit $gapMin$ la valeur fixée de la contrainte *gap minimum*, $gapMax$ la valeur fixée de la contrainte *gap maximum* et $winMax$ la valeur fixée de la contrainte de *fenêtre de temps maximum*. Soit un motif $z = \alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \rightarrow \alpha_n$.

- Une occurrence de z vérifie la contrainte du gap minimum si et seulement si, pour tout α_i, $i = 1 \ldots n-1$, $eid(\alpha_{i+1}) - eid(\alpha_i) \geq gapMin$.

- Une occurrence de z vérifie la contrainte du gap maximum si et seulement si, pour tout α_i, $i = 1 \ldots n-1$, $eid(\alpha_{i+1}) - eid(\alpha_i) \leq gapMax$.

- Une occurrence de z vérifie la contrainte de fenêtre maximum si et seulement si $eid(\alpha_n) - eid(\alpha_1) \leq winMax$.

Exemple 2 *Par soucis de clarté, nous allons illustrer les contraintes de temps présentées ci-dessus individuellement et en ne considérant que la première séquence de la base exemple de la figure 2.1.*

- Supposons que la contrainte de fenêtre de temps maximum ait été fixée à 3. Dans ce cas, l'occurrence A(1) $\rightarrow$ BC(2) $\rightarrow$ F(5) du motif $A \rightarrow BC \rightarrow F$ ne vérifie pas la contrainte $winMax = 3$ car $eid(F) - eid(A) = 5 - 1 = 4 > winMax$, alors que l'occurrence BC(2) $\rightarrow$ A(3) $\rightarrow$ D(4) du motif $BC \rightarrow A \rightarrow D$ la vérifie puisque $eid(D) - eid(BC) = 4 - 2 = 2 \leq winMax$. Dans le cas de séquences d'achat, l'utilisation d'une telle contrainte assure aux utilisateurs que les motifs extraits ne porteront pas sur des périodes de temps trop longues.

- Supposons que la contrainte de gap minimum ait été fixée à 2. Dans ce cas, l'occurrence B(2) $\rightarrow$ D(4) $\rightarrow$ C(5) du motif $B \rightarrow D \rightarrow C$ ne vérifie pas la contrainte à cause du laps de temps trop court qu'il y a entre se deux derniers événements ($eid(C) - eid(D) = 5 - 4 = 1 < gapMin$). En revanche, l'occurrence A(1) $\rightarrow$ AC(3) $\rightarrow$ F(5) du motif $A \rightarrow AC \rightarrow F$ respecte cette contraintes car $eid(F) - eid(AC) = 5 - 3 = 2 = gapMin$ et $eid(AC) - eid(A) = 3 - 1 = 2 = gapMin$. On peut noter que le motif $A \rightarrow AC \rightarrow F$ possède plusieurs autres occurrences dans cette séquence (A (1) $\rightarrow$ AC (2) $\rightarrow$ F(5) ou encore A (2) $\rightarrow$ AC (3) $\rightarrow$ F(5)), et qu'aucune de ces occurrences ne respecte la contrainte. Ainsi, l'occurrence présentée est la seule occurrence permettant de vérifier la contrainte, ce qui est suffisant pour affirmer que le motif respecte la contrainte. Dans le cas de séquences d'achats, cela permet d'extraire des motifs dont les successions d'achats ne sont pas trop rapprochées.

- Supposons que la contrainte de gap maximum soit fixée à 2. Dans ce cas, l'occurrence A(1) $\rightarrow$ B(2) $\rightarrow$ CF(5) du motif $A \rightarrow B \rightarrow CF$ ne vérifie pas la

contrainte puisque $eid(CF) - eid(B) = 5 - 2 = 3 > gapMax$. *Par contre, l'occurrence B(2)* $\rightarrow$ *C(3)* $\rightarrow$ *D(4) du motif* $B \rightarrow C \rightarrow D$ *la vérifie puisque* $eid(D) - eid(C) = eid(C) - eid(B) = 1 < gapMax$. *Toujours dans le cas de séquences d'achats, l'utilisation de la contrainte de gap maximum évite l'extraction de motifs dont les transactions seraient trop éloignées les unes des autres.*

Une autre contrainte de temps permet de regrouper plusieurs transactions pour n'en former qu'une seule. Cette contrainte, appelée *durée de rectification* dans [SA96], permet par exemple de considérer que les achats effectués sur plusieurs jours ne soient considérés que comme une seule et unique transaction.

Exemple 3 *Afin de fixer le mécanisme de la durée de rectification, reprenons la même séquence que dans l'exemple précédent, avec une valeur de gap maximum fixée à 2, une durée de rectification fixée à 2 et considérons le motif* $A \rightarrow B \rightarrow CF$. *L'occurrence A(1)* $\rightarrow$ *B(2)* $\rightarrow$ *CF(5) de ce motif ne respectait pas la contrainte de gap maximum car* $eid(CF) - eid(B) = 5 - 2 = 3 > gapMax$. *La durée de rectification permet de regrouper les événements D et CF de la séquence pour former un seul événement dont la date correspond à la date de l'événement D, c'est-à-dire 4. Dans ces conditions, l'occurrence précédente vérifie la contrainte de gap maximum puisque maintenant* $eid(CF) = 4$ *et donc* $eid(CF) - eid(B) = 4 - 2 = 2 = gapMax$.

2.2.3 Les contraintes syntaxiques

Ce type de contraintes permet de spécifier la structure des motifs extraits. Ainsi, les experts peuvent souhaiter ne s'intéresser qu'aux motifs ne portant que sur une succession limitée d'événements (*contrainte de longueur*) dont la largeur peut être limitée (*contrainte de largeur*). Dans certains domaines d'application, la présence ou l'absence de tel ou tel item dans les motifs extraits peut être utile. Par exemple, dans le domaine de l'analyse de séquences d'achats, il est possible de ne pas considérer l'item 'béret basque' comme important pour la suite d'une étude. Dans ce cas, il est possible de spécifier une *contrainte d'exclusion* assurant aux motifs extraits de ne pas inclure certains items ou inversement, en spécifiant une *contrainte d'inclusion* qui assure aux motifs extraits de posséder au moins un événement incluant la présence de tel ou tel item.

Exemple 4 - *Dans le cas d'une contrainte de longueur fixée à 3, le motif* $ABC \rightarrow C \rightarrow D$ *ne vérifie pas la contrainte puisque le ce motif est un 5-motif. De même, si une contrainte de largeur a été fixée à 2, alors ce même motif ne sera pas correct puisque son premier événement (ABC) a une largeur de 3.*

- *En appliquant une contrainte d'inclusion sur l'item A, le motif* $BC \rightarrow D \rightarrow F$ *ne respecte pas cette contrainte, puisqu'aucun de ses événements ne comporte l'item A. De même, si l'on applique une contrainte d'exclusion portant sur l'item*

C, dans ce cas ce motif ne sera pas correct puisque son premier événement BC contient l'item exclu.

Dans [CMB02], un algorithme permettant de gérer une contrainte de similarité avec un motif de référence a été présenté. Cette contrainte est proche d'une contrainte syntaxique plus connue, utilisée par exemple dans [GRK99, PHW02, ALB03], qui impose aux motifs extraits d'appartenir au langage décrit par une expression régulière. Par exemple, dans [PHW02], il est possible d'imposer que les motifs extraits commencent par l'item A et comportent au moins un événement incluant l'item B ou l'item C. Dans ce cas, les motifs $B \rightarrow C$ ne satisfait pas la contrainte alors que le motif $A \rightarrow A \rightarrow C$ la vérifie.

2.2.4 Les autres contraintes

Plus récemment, un autre type de contraintes a été présenté et étudié afin d'être intégré à un processus d'extraction de motifs séquentiels sous contraintes ([PHW02]). Ces contraintes, appelées *contraintes agrégats*, portent sur le contenu plus spécifique des événements constituant les motifs. Ainsi, il est possible de limiter la somme totale ou la moyenne de tous les événements constituant les motifs. Par exemple, dans les domaines de l'analyse de séquences d'achats, les utilisateurs peuvent ne s'intéresser qu'aux motifs dont le total des transactions couvre une fourchette de prix fixée ou encore dont la moyenne des achats ne dépasse pas une certaine valeur.

D'autres contraintes ont aussi été utilisées, comme par exemple dans l'algorithme SLPMiner [SK02], qui exploite une contrainte de fréquence dont la valeur décroît lorsque la longueur des motifs augmente (i.e., le seuil de fréquence d'extraction devient plus petit pour des motifs plus grands).

2.2.5 Propriété et prise en compte des contraintes

L'utilisation de ces différentes contraintes permet à l'utilisateur d'obtenir des motifs qu'il juge plus particulièrement intéressants. Toutefois, l'intérêt des contraintes va bien au delà de la sélection des motifs, puisque leur utilisation active peut permettre de réduire significativement l'espace de recherche et de gagner ainsi en efficacité. En effet, celle-ci a pour effet d'aller rapidement vers les sous-espaces contenant les motifs qui satisferont aux contraintes et d'éviter d'explorer certains sous-espaces qui n'ont aucune chance de contenir des motifs les satisfaisant.

Les algorithmes existants exploitent différentes propriétés de ces contraintes. Les principales propriétés identifiées sont la *monotonie* et l'*anti-monotonie*. Une contrainte est dite *anti-monotone* si lorsqu'une séquence ne satisfait pas la contrainte, alors aucune séquence la contenant ne la satisfera. Une contrainte est dite *monotone* si lorsqu'une séquence ne la satisfait pas, alors aucune de ses sous-séquences ne la satisfera. Par exemple, pour la contrainte de fréquence minimum,

qui est une contrainte anti-monotone, si on sait que le motif A $\rightarrow$ B n'est pas fréquent, alors le sous-espace formé par les motifs contenant A $\rightarrow$ B peut être ignoré lors de la recherche. Parmi les contraintes anti-monotones, on peut citer, en plus de la contrainte de fréquence minimum, la contrainte d'absence d'un item donné dans les motifs ou encore la contrainte de longueur et de largeur maximum. La catégorie des contraintes monotones, quant à elle, contient notamment la contrainte de présence d'un item donné, la contrainte de fréquence maximale ou encore les contraintes de longueur et de largeur minimum. Il existe des techniques qui permettent de gérer des contraintes qui sont ni monotones, ni anti-monotones. Par exemple, pour les expressions régulières, on peut citer les algorithmes développés dans SPIRIT [GRK99].

La plupart des algorithmes d'extraction de motifs séquentiels sous contraintes qui exploitent des combinaisons de plusieurs contraintes, le font pour des combinaisons fixées et utilisent des approches sophistiquées spécifiques. Par exemple, on peut citer l'adaptation dynamique de la stratégie de prise en compte d'une contrainte de fréquence minimum et d'une contrainte d'expression régulière proposée par [ALB03].

Récemment, certains travaux (e.g.,[PHW02]) ont eu pour objectif d'étudier des stratégies générales pour l'exploitation de certaines combinaisons de contraintes. Ces travaux vont dans le sens de travaux similaires effectués dans le domaine des données non séquentielles (e.g., [Jeu02, BGMP03a, BGMP03b]).

2.3 Les différentes familles d'algorithmes

Un certain nombre d'algorithmes ont été proposés permettant de couvrir un large éventail d'applications et de contraintes. Tous ces algorithmes possèdent leurs spécificités mais d'une manière générale, il est possible de mettre en avant deux étapes communes, indispensables à un processus d'extraction de motifs séquentiels : une étape de *génération des candidats* et une étape de *comptage des supports*.

La première étape consiste à générer les successions d'événements qui sont susceptibles d'apparaître dans les séquences de la base et la deuxième permet de tester la fréquence des motifs candidats générés. Dans la majeure partie des cas, la contrainte de fréquence minimum reste utilisée en complément à l'utilisation de nouvelles contraintes, laissant l'étape de comptage omniprésente lors d'un processus d'extraction. Chacune de ces étapes est effectuée différemment suivant les algorithmes, et les spécificités développées par chacun d'eux permettent de les classer en trois familles d'algorithmes distinctes.

Dans cette section nous présentons chacune de ces familles en essayant de mettre en avant leurs particularités communes et en développant pour chacune d'elles un algorithme choisi comme représentant de cette famille. Nous développons ainsi plus en détails les algorithmes GSP [SA96], PrefixSpan [PHMAP01] et *Spade* [Zak01, Zak98] ainsi que sa version contrainte *cSpade* [Zak00]. Ces deux derniers

algorithmes font l'objet d'un développement plus spécifique car les algorithmes proposés dans le chapitre suivant (chapitre 3) appartiennent à cette famille.

2.3.1 Approche type *Apriori*

La première famille regroupe les algorithmes qui effectuent une passe complète sur la base afin de calculer le support des candidats ([AS95, SA96, MCP98, MPT99, Mas02, GRK99, MTV95]). Ces approches permettent d'assurer un nombre de passes limité, exactement égal à la taille du plus long motif fréquent, sur une base lors d'une extraction. Comme nous l'avons déjà mentionné dans l'introduction, la génération par niveau et l'exploitation de l'anti-monotonicité de la fréquence ont été ultérieurement utilisés dans un algorithme baptisé APriori ([AMS+96, AS94a]), bien connu dans le domaine de l'extraction de motifs fréquents dans des données sans composante temporelle. Par référence à ces premiers travaux, nous avons nommé cette famille d'approches *famille APriori*. On notera que le principe de parcours par niveau (i.e., en largeur), tirant partie de l'anti-monotonicité d'un critère de sélection (ici la fréquence), a été formulé dans un cadre plus général dans [MT97].
L'algorithme WINEPI [MTV95], contrairement aux autres algorithmes de cette famille, effectue des extractions de motifs séquentiels à partir d'une seule séquence et utilise une notion de fenêtre glissante pour évaluer la fréquence des motifs.

Parmi les approches de cette famille, un effort particulier a été fait afin de rendre plus efficace le comptage des candidats (par exemple, l'arbre de hachage utilisé dans [SA96], développé dans la partie suivante), et éviter la génération de candidats inutiles (par exemple, les candidats ne vérifiant pas les contraintes syntaxiques [GRK99, ALB03]).

Le principe général est représenté par l'algorithme 1 et se déroule de la façon suivante :

- Une première passe est effectuée sur la base initiale afin d'identifier les items fréquents. Cette étape donne lieu à la construction de l'ensemble F_1 contenant tous les 1-motifs fréquents, c'est-à-dire tous les items i de I dont le support est supérieur au support minimum.

- Ensuite, pour les motifs de taille k ($k \geq 1$) l'algorithme effectue les opérations suivantes :
→ GENERATION_CANDIDATS : cette étape correspond à la création des motifs candidats de longueur $k+1$ (C_{k+1}) à partir de l'ensemble des motifs fréquents de longueur k (F_k). Pour ce faire, le processus considère deux à deux tous les motifs fréquents appartenant à l'ensemble F_k afin de générer l'ensemble des motifs candidats de taille $k+1$. Par exemple, l'algorithme [MCP98] va générer un nouveau motif de taille $k+1$ à partir de deux motifs de longueur k si ils partagent un même

préfixe de taille $k-1$
$\rightarrow$ COMPTER_SUPPORTS : cette étape permet d'identifier les motifs candidats effectivement fréquents. Pour cela, le processus va effectuer une passe sur la base et va incrémenter les supports de tous les motifs candidats contenus dans C_{k+1}.

Algorithme 1 (Approche type APriori)
Entrée : *Une base de séquences et un support minimum minSup*
Sortie : F, *ensemble de tous les motifs séquentiels fréquents*

$F_1 = \{i \in I | support(i) \geq minSup\}$
$k = 1$
1. **while** $F_k \neq \emptyset$ **do**
2. C_{k+1} := *GENERATION_CANDIDATS*(F_k)
3. *COMPTER_SUPPORTS*(C_{k+1})
4. $F_{k+1} := \{s \in C_{k+1} \mid support(s) \geq minSup\}$
5. $k := k+1$
6. **od**
7. **output** $F = \bigcup_{1 \leq j < k} F_j$

Exemple 5 *La figure 2.2 illustre le déroulement partiel d'une extraction effectuée sur la base présentée par la figure 2.1, avec un support minimum fixé à 50%. Chaque niveau de l'arbre représente une partie des motifs candidats de taille k (C_k) accompagnés de leur support. Ces supports sont calculés à chaque niveau par une passe complète de la base. Les motifs grisés correspondent aux motifs candidats reconnus comme étant non fréquents après l'étape de comptage des supports. La génération des motifs candidats au niveau $k+1$ s'effectue en considérant deux à deux les motifs non grisés du niveau k. Ces motifs correspondent à des motifs fréquents, c'est-à-dire appartenant à F_k. La génération d'un nouveau motif candidat au niveau $k+1$ s'effectue à partir de deux motifs fréquents de taille k qui partagent un préfixe commun de taille $k-1$. Par exemple, le motif $AB \rightarrow B$ au niveau $k=3$ a été généré à partir des motifs AB et $A \rightarrow B$, fréquents au niveau $k=2$ et partageant le préfixe commun A. Plusieurs techniques de génération différentes ont été proposées dans la littérature afin de garantir la complétude de la génération des candidats. Nous en décrivons une particulière dans la section suivante (algorithme GSP).*

L'algorithme GSP

Dans cette partie, nous allons décrire de façon plus spécifique l'algorithme GSP (Generalized Sequential Patterns) présenté dans [SA96] car celui-ci est tout

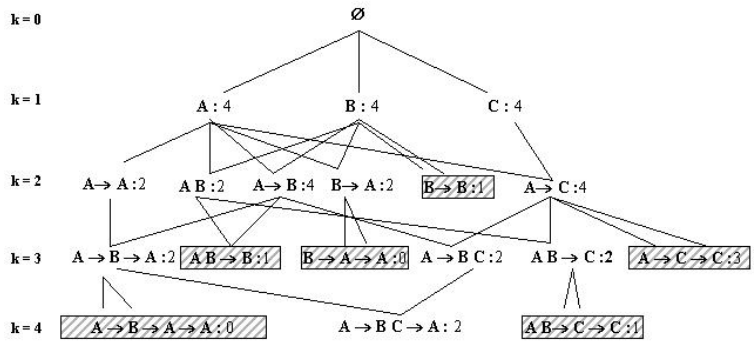


FIG. 2.2 – Exemple de candidats générés lors de différentes itérations de l'algorithme 1 (parcours en largeur).

à fait représentatif de la famille présentée précédemment. Les deux algorithmes précurseurs de GSP sont les algorithmes APrioriAll et APrioriSome présentés dans [AS95]. Ces derniers ne prenaient pas en compte d'autres contraintes que la contrainte de fréquence minimum, introduites par la suite dans GSP. Dans cette partie, nous nous limiterons à une explication de l'algorithme GSP sans faire intervenir une autre contrainte que la contrainte de fréquence minimum. L'aspect contraint sera développé dans la section suivante.

Les spécificités de l'algorithme GSP reposent d'une part sur le mode de génération des candidats, et d'autre part sur la structure particulière utilisée pour représenter ces candidats et sur laquelle se base tout le processus d'extraction.

Pour GSP, la génération des candidats diffère de la génération utilisée dans l'exemple précédent (génération par préfixe commun), et s'effectue de la façon suivante : deux motifs fréquents de longueur k, z_1 et z_2 vont donner lieu à la génération d'un nouveau motif de longueur $k+1$, si et seulement si le motif obtenu en supprimant le premier item de z_1 est identique au motif obtenu en supprimant le dernier item de z_2. Par exemple, le motif AB $\rightarrow$ D $\rightarrow$ C est généré à partir des motifs AB $\rightarrow$ D (z_1) et B $\rightarrow$ D $\rightarrow$ C (z_2). De même, pour le motif A $\rightarrow$ BC, z_1 et z_2 correspondent respectivement aux motifs A $\rightarrow$ B et BC.

La génération des motifs de longueur 2 s'effectue d'une façon particulière puisqu'il est impossible de retirer un item des motifs de longueur 1. Celle-ci s'effectue alors de la façon suivante : pour tout couple (z_1,z_2) provenant de F_1 (ensemble des items fréquents), si $z_1 = z_2$ alors GSP va générer le candidat $z_1 \rightarrow z_2$ et si $z_1 \neq z_2$, alors deux motifs vont être générés : le motif $z_1 z_2$ et le motif $z_1 \rightarrow z_2$. Par exemple, à partir du couple d'items (A,A), GSP va générer un seul motif, A $\rightarrow$ A, et deux motifs à partir du couple (A,B) : AB et A $\rightarrow$ B.

La seconde spécificité de GSP repose sur la structure utilisée pour ranger les motifs candidats après chaque étape de génération. Cette structure correspond à un arbre de hachage qui permet de retrouver les candidats appartenant à une séquence de façon plus efficace. La figure 2.3 donne un exemple d'un arbre de hachage pour les motifs candidats de longueur 3 (qui correspond à l'ensemble C_3).

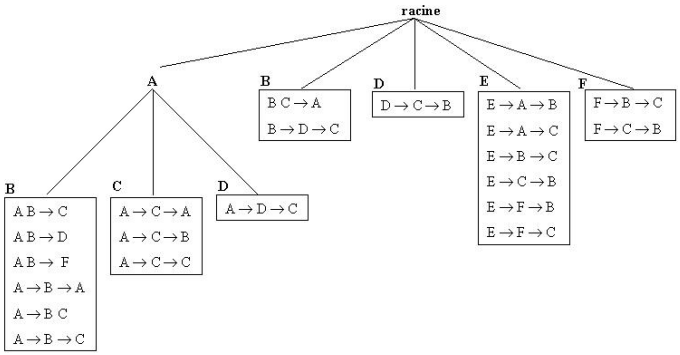


FIG. 2.3 – La structure hash-tree de GSP.

Les motifs candidats sont ajoutés dans l'arbre de hachage de la façon suivante : GSP parcourt tous les items d'un motif candidat donné et va descendre dans l'arbre de hachage. Lorsque GSP arrive à une feuille, soit le candidat est ajouté à cette feuille, soit la feuille est scindée en deux ou en plusieurs nouvelles feuilles dans lesquelles les candidats sont répartis. Cette deuxième possibilité apparaît lorsque le nombre de candidats présents dans une feuille dépasse une certaine valeur fixée au préalable.

Une fois que les motifs candidats ont été générés, GSP procède à l'identification de tous les motifs candidats susceptibles d'appartenir à une séquence donnée. Pour une séquence s, tous les items de s sont considérés un à un et la fonction de hachage est appliquée sur chacun d'eux, à partir de la racine de l'arbre, afin d'aller rechercher tous les motifs candidats commençant par cet item. Cette approche permet de s'assurer qu'aucun candidat ne commençant pas sur un item de la séquence ne sera considéré. Pour une occurrence d'un item i de s ayant permis d'atteindre un noeud n, fils de la racine, le traitement se répète. L'algorithme poursuit alors sa descente vers les noeuds fils de n en réalisant un hachage sur chacune des occurrences suivant l'occurrence de i considérée dans s. Lorsqu'une feuille de l'arbre est atteinte, les candidats de cette feuille sont ajoutés dans une liste de motifs candidats susceptibles d'être contenus dans s. Enfin, une fois que tous les motifs candidats ont été identifiés, GSP va vérifier qu'ils appartiennent

bien à la séquence courante et va auquel cas incrémenter leur support.

Par exemple, considérons l'arbre de hachage de la figure 2.3, et une séquence s = A $\rightarrow$ C $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C. Afin de constituer la liste, notée $\mathcal{L}$, des motifs candidats susceptibles d'appartenir à cette séquence, le processus va commencer par appliquer la fonction de hachage sur l'item A, à partir de la racine de l'arbre. A partir de ce noeud, le hachage est effectué sur tous les items qui le suivent dans s, c'est-à-dire les items C, B, D et C. Le traitement sur C permet d'atteindre la feuille contenant les motifs A $\rightarrow$ C $\rightarrow$ A, A $\rightarrow$ C $\rightarrow$ B et A $\rightarrow$ C $\rightarrow$ C qui sont alors ajoutés à $\mathcal{L}$. Le hachage sur le deuxième item suivant A, c'est-à-dire l'item B permettra d'ajouter les motifs AB $\rightarrow$ C, AB $\rightarrow$ D, AB $\rightarrow$ F, A $\rightarrow$ B $\rightarrow$ A, A $\rightarrow$ BC et A $\rightarrow$ B $\rightarrow$ C. Le traitement sur D ajoutera le motif A $\rightarrow$ D $\rightarrow$ C, et enfin, le traitement sur C n'est pas nécessaire puisqu'il a déjà été effectué. Ensuite, le traitement reprend à partir de la racine avec le $2^{éme}$ item de s, c'est-à-dire l'item C. Celui-ci n'apparaît pas comme fils de la racine. Le traitement se poursuit alors sur B, le $3^{éme}$ item de s. Le hachage selon cet item permet d'ajouter les motifs BC $\rightarrow$ A et B $\rightarrow$ D $\rightarrow$ C à la liste $\mathcal{L}$, et enfin, le traitement des derniers items de s permet encore de collecter le motif D $\rightarrow$ C $\rightarrow$ B. La liste de motifs ainsi obtenue contient des motifs qui doivent encore être testés afin de vérifier qu'ils apparaissent effectivement dans s. Si tel est le cas, leur support est incrémenté.

Contraintes

L'algorithme GSP permet d'extraire de motifs séquentiels sous contraintes en considérant des contraintes de temps. Ces contraintes correspondent aux contraintes de gap minimum et maximum ($maxGap$) ainsi qu'à la contrainte de rectification (notée $windowSize$, cf. section 2.2.2). La prise en compte de ces contraintes intervient lors de la recherche des candidats susceptibles d'appartenir à une séquence et lors de la vérification des motifs candidats appartenant effectivement à une séquence.

Recherche des motifs candidats appartenant à une séquence :
Pour cela, GSP explore l'arbre de hachage à partir de la racine. Plusieurs situations se présentent alors :
- au niveau de la racine : GSP va appliquer la fonction de hachage de façon récursive sur chaque item rencontré dans la séquence de données.
- au niveau d'un noeud : supposons que l'item atteint appartienne à un événement apparaissant à une date t. L'algorithme GSP va alors appliquer la fonction de hachage sur chaque item de la séquence appartenant aux événements présents dans l'intervalle $[t - windowSize,\ t + max(windowSize, maxGap)]$. Cela permet de s'assurer que les occurrences considérées vérifieront bien la contrainte de gap maximum en considérant la fenêtre de rectification.
- au niveau d'une feuille : lorsqu'une feuille est atteinte, GSP va considérer tous

les motifs candidats contenus dans cette feuille. Pour chacun de ces motifs, GSP va vérifier qu'il apparaît bien dans la séquence considérée.

Vérification de la présence d'un motif candidat dans une séquence :

Afin de vérifier qu'un motif candidat apparaît bien dans une séquence de données, GSP va alors appliquer un processus en deux phases : phase avant et phase arrière. La phase avant consiste à retrouver le maximum d'items successifs du motif candidat dans la séquence de données. Lors de cette phase avant, le processus s'assure que la contrainte de gap minimum est bien vérifiée entre tous les événements du motif. Dès qu'un item du motif candidat n'apparaît pas dans la séquence de données, ou qu'il n'existe pas d'occurrence de cet item permettant de vérifier la contrainte de gap minimum, alors le processus de vérification s'arrête pour ce candidat. Maintenant, si un item considéré apparaît dans la séquence mais que le gap qui s'écoule avec la date d'apparition du dernier item vérifié est supérieur à la valeur du gap maximum fixé, alors le processus bascule sur la phase arrière, ce qui permet de remonter dans les items vérifiés jusqu'à ce que l'item concerné puisse vérifier la contrainte $gapMax$.

2.3.2 Approche par projections

L'approche par projections a été initialement proposée dans le cadre de l'extraction de motifs fréquents dans des données sans composante temporelle ([HPY00, AAP01]), puis a été étendue au domaine des motifs séquentiels. Les algorithmes appartenant à cette deuxième famille ([HHMAC+00, PHMAP01, PHW02, GGK01]) s'appuient sur les coûts que représentent la génération et le comptage du support des motifs candidats pour les algorithmes de la première famille. Ainsi, ils cherchent d'une part à réduire la taille des bases à considérer lors de l'étape de comptage des supports, et d'autre part à réduire l'espace de recherche en supprimant la génération de candidats lorsque ceux-ci n'existent pas dans les séquences.

Pour ce faire, ils travaillent sur des bases projetées qui correspondent à des sous-ensembles de la base de séquences initiale. Ces bases projetées ont l'avantage d'avoir une taille réduite par rapport à la base initiale, réduisant ainsi le coût lié à l'étape de comptage des supports. Par exemple, l'algorithme PrefixSpan [PHMAP01] emploie une projection des bases selon un préfixe donné (cf. figure 2.4). De plus, ces bases projetées ne contiennent que des items susceptibles d'appartenir à des motifs fréquents, ce qui permet d'économiser sur la génération de motifs candidats qui n'ont pas d'existence dans la base.

L'algorithme 2 donne un schéma simplifié illustrant le principe général des algorithmes de cette famille. Celui-ci ne couvre que le cas de génération de motifs ne contenant qu'un seul item par événement, c'est-à-dire de la forme $\alpha_1 \rightarrow \alpha_2 \rightarrow \ldots \alpha_n$ où chaque événement α_i ($i \in [1,n]$) n'est composé que d'un seul item. Ce processus est récursif et comprend trois étapes :

- 1ère étape : cette étape identifie les items fréquents, correspondant aux motifs de taille 1, en effectuant une passe complète sur la base initiale.

- 2ème étape : chaque motif fréquent de taille 1 identifié lors de la 1ère étape, donne lieu à la création d'un sous-ensemble de la base initiale, appelé *base projetée*. Chaque base projetée ainsi créée va contenir un sous-ensemble de tous les motifs fréquents et sera traitée individuellement et récursivement dans le reste du processus.

- 3ème étape : une passe est effectuée sur chacune des bases projetées créées lors de la 2ème étape afin d'identifier les items fréquents dans cette nouvelle base. Le processus réitère la création de la base projetée et l'identification des items fréquents dans cette base, sur chacun des items fréquents, permettant la découverte de motifs fréquents de plus en plus longs. Le processus s'arrête lorsque plus aucun item fréquent n'est identifié.

Algorithme 2 (Approche par projections)
Entrée : *Une base de séquences D_z correspondant à une base projetée suivant un motif z, le motif courant z et un support minimum minSup*
Sortie : *F, ensemble de tous les motifs séquentiels fréquents*
Premier appel : *GENERATION(D,$\emptyset$,minSup)*
Appel récursif : *GENERATION($D_{|z}$,z,minSup)*
Fonction appelée : *PROJECTION($D_{|z}$,z'). Effectue la projection de la base $D_{|z}$ suivant le motif z'*

$F := \emptyset$
$suffixeFreq = \{i \in I | support(i) \geq minSup \text{ sur } D_{|z}\}$
1. **for all** $i \in suffixeFreq$ **do**
2. $z' := z \rightarrow i$
3. $D_{|z'} := PROJECTION(D_{|z}, z')$
4. $F := F \cup \{ z' \} \cup GENERATION(D_{|z'}, z', minSup)$
5. **od**
6. **output** F

L'algorithme PrefixSpan

PrefixSpan travaille sur des bases projetées dont le représentant correspond au préfixe des motifs fréquents qui en découlent. La génération d'un motif de longueur k s'effectue en ajoutant un item à la fin d'un préfixe de longueur $k-1$. Les

bases ainsi projetées sont réduites à des sous-séquences des séquences initiales qui débutent sur les premier item qui suit le dernier item du préfixe courant. Afin de faciliter la compréhension de PrefixSpan, nous limitons nos explications aux cas de motifs composés d'un seul item par événement.

PrefixSpan applique un parcours en profondeur de l'espace des motifs candidats. La figure 2.5 représente le sous-arbre des motifs candidats pour le préfixe A. Il correspond à l'espace de génération construit lors d'une extraction effectuée sur la base de la figure 2.1 avec un seuil de fréquence minimum fixé à 2 séquences (i.e., 50% des séquences de la base). Sur cet arbre, chaque noeud représente un motif généré suivi de son support et les noeuds grisés correspondent à des motifs non fréquents. A partir de chaque noeud non grisé, c'est-à-dire pour chaque motif fréquent z, PrefixSpan va construire une base projetée qui permettra de générer tous les motifs candidats z' ayant pour préfixe z. L'arbre des candidats est parcouru en profondeur, c'est-à-dire que PrefixSpan va traiter tous les sous-arbres débutant sur un motif fréquent jusqu'à ce qu'il atteigne une feuille (motif non fréquent ou aucun item fréquent dans la base projetée). L'ordre de parcours effectué sur ce sous-arbre est précisé par le chiffre indiqué à côté de chaque arc. Ainsi, à partir du préfixe A, le parcours en profondeur appliqué par PrefixSpan est le suivant : A $\rightarrow$ A, A $\rightarrow$ B, A $\rightarrow$ B $\rightarrow$ A, A $\rightarrow$ B $\rightarrow$ C, A $\rightarrow$ C,...

La figure 2.4 donne un exemple des bases projetées obtenues à partir des préfixes A,B,C, A$\rightarrow$A,A$\rightarrow$C, A$\rightarrow$C$\rightarrow$A et A $\rightarrow$ C $\rightarrow$ B. La première colonne correspond au préfixe courant dont la base projetée est représentée dans la deuxième colonne. Les deux dernières colonnes représentent respectivement les items ayant été identifiés comme fréquents dans la base projetée, ainsi que les motifs séquentiels générés à partir de ces items. Le support associé à chacun d'eux est calculé lors d'une passe sur la base projetée correspondante. Ainsi, la construction de chaque ligne du tableau nécessite d'effectuer deux passes dans la base. Lors d'une première passe, la base projetée est construite et les items fréquents identifiés et lors de la deuxième passe, le support des motifs séquentiels générés est calculé. Les motifs séquentiels en gras mettent en évidence les motifs dont le support respecte le seuil minimum fixé ($\geq$ 2) et sur lesquels la récursion va s'appliquer. Ainsi, le traitement de tous les motifs du sous-arbre de A entraîne la construction de 11 bases projetées.

Contraintes

Récemment, Jian Pei et al. ont présentés une version contrainte de PrefixSpan ([PHMAP01]) permettant de prendre en compte un large éventail de contraintes (e.g., contraintes syntaxiques et contraintes de temps) et en introduisant la gestion d'un type de contraintes peu exploitées jusqu'à présent, les contraintes d'agrégats.

Cette approche, présentée dans [PHW02], introduit une classe de contraintes plus générale qui possède des propriétés compatibles avec l'approche utilisée par

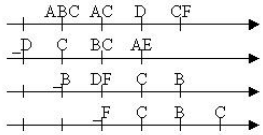
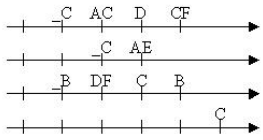
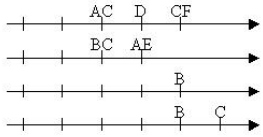
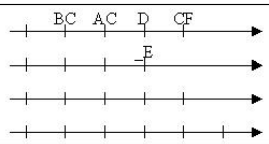
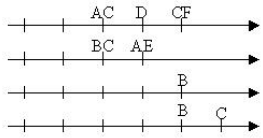
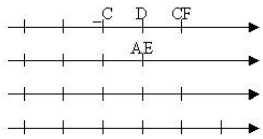
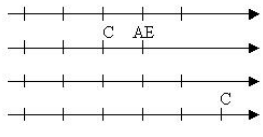
Préfixes	Bases projetées	Items Fréquents	Motifs séquentiels
A		A:2 B:4 C:4 D:3 F:3	**A** →**A** :2; **A** → **B** :4; **A** → **C** :4; **A** →**D** :2 ; **A** → **F** :2
B		A:2 C:4 D:3 F:2	**B** →**A** :2; **B** → **C** :3; **B** → **D** :2; **B** →**F** :2 ;
C		A:2 B:3 C:3	**C** →**A** :2 ; **C** → **B** :3; **C** → **C** :3;
A→A		∅	∅
A→C		A:2 B:3 C:3	**A** →**C** →**A** :2 **A** →**C** → **B** :3 **A** →**C** → **C** :3
A→C→A		∅	∅
A→C→B		C:2	A→C→B →C:1

FIG. 2.4 – Exemple de bases projetées avec PrefixSpan.

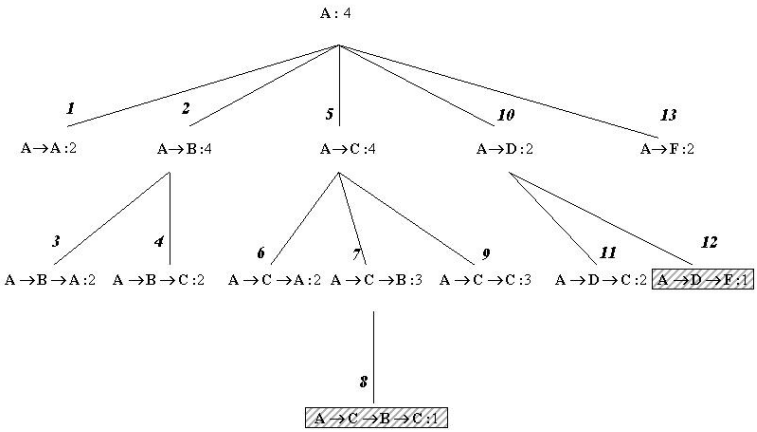


FIG. 2.5 – Exemple d'un arbre de parcours en profondeur.

PrefixSpan. Cette classe, que l'on appellera *préfixe-stables*, regroupe aussi bien les contraintes anti-monotones que les contraintes monotones vis-à-vis du préfixe. Une contrainte est une *contrainte anti-monotone vis-à-vis du préfixe* si lorsqu'un motif séquentiel α de la forme $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_n$ satisfait la contrainte, alors tous les motifs séquentiels de la forme $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_i$ avec $i \in [1,n]$ respectent aussi la contrainte (e.g., la contrainte de gap maximum et la contrainte de longueur maximum). Une contrainte est une *contrainte monotone vis-à-vis du préfixe* (e.g., la contrainte de présence d'items particuliers et la contrainte de longueur minimum) si lorsqu'un motif séquentiel α vérifie la contrainte, alors celle-ci sera aussi vérifiée par tous les motifs commençant par α (e.g., $\alpha \rightarrow$ ...).

L'intégration de contraintes *anti-monotones vis-à-vis du préfixe* peut être effectuée sans avoir à apporter trop de modifications majeures à l'approche utilisée dans PrefixSpan. Cependant, celle-ci devient plus délicate pour les contraintes *monotones vis-à-vis du préfixe* et pour certaines contraintes d'agrégats (e.g., *moyenne*). Dans cette partie, nous expliquons les principales modifications apportées à PrefixSpan dans le cas général de présence de contraintes appartenant à la classe préfixe-stables et dans le cas particulier de présence de certaines contraintes d'agrégats.

Gestion de contraintes monotones vis-à-vis du préfixe :
Comme il a été précisé précédemment, les contraintes anti-monotones vis-à-vis du préfixe, peuvent être introduites sans trop de difficultés majeures. Cependant, la prise en compte des contraintes monotones vis-à-vis du préfixe est plus délicate. Par exemple, considérons une contrainte de somme minimum (*sum*) fixée à 50, et une base de séquences composées d'items ayant une valeur positive. Dans un tel contexte, il n'est pas possible de supprimer la récursion à partir de l'item ayant une

valeur de 25, même si ce motif ne vérifie pas la contrainte. En effet, il est possible de générer un motif ayant l'item 25 comme préfixe et qui vérifie la contrainte (e.g., le motif 25 $\rightarrow$ 30 est tel que 25+30=55 $> sum$).

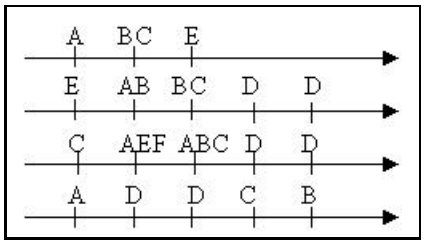


FIG. 2.6 – Base exemple pour PrefixSpan avec gestion des contraintes.

Nous illustrons ces modifications sur le cas particulier d'une extraction incluant une contrainte d'expression régulière afin de fixer le principe. Celle-ci n'est pas une contrainte préfixe-stable en tant que telle, mais la contrainte 'un motif est **légal** pour une expression régulière', définie ci-dessous, est préfixe stable.

Considérons la base de la figure 2.6, et fixons une contrainte de seuil de fréquence minimum à 2 et définissons la contrainte d'expression régulière de la façon suivante : A$*${B$\rightarrow$B|BC$\rightarrow$D|D$\rightarrow$D}. Cette contrainte signifie que les motifs extraits doivent avoir l'une des formes suivantes : A $\rightarrow \ldots \rightarrow$ B $\rightarrow$ B, A $\rightarrow \ldots \rightarrow$ BC $\rightarrow$ D ou A $\rightarrow \ldots \rightarrow$ D $\rightarrow$ D.

Comme pour PrefixSpan, le processus commence par effectuer une passe dans la base afin d'identifier les items fréquents. Lors de cette étape, les items non fréquents ainsi que les items n'apparaissant pas dans l'expression régulière sont supprimés. Ainsi, l'item F est supprimé de la base puisqu'il est infréquent et les items A, B, C, D et E sont identifiés comme des 1-motifs fréquents. Lors de cette même passe, les séquences ne comportant aucun motif représenté par l'expression régulière sont supprimés et ne seront plus considérées dans la suite du processus. Ainsi, la première séquence A $\rightarrow$ BC $\rightarrow$ E est supprimée de la base puisque celle-ci ne vérifie pas à l'expression régulière.

Dans la suite du processus, seuls les préfixes susceptibles de générer un motif qui puisse satisfaire la contrainte d'expression régulière sont traités. Ainsi, la récursion ne s'appliquera pas sur les 1-motifs B,C,D et E puisque qu'ils ne généreront pas de motifs satisfaisant l'expression régulière. Seul le 1-motif A sera utilisé comme préfixe valable dans la suite du processus.

Le processus s'arrête lorsque les différentes bases projetées ne contiennent plus d'items fréquents, ou encore lorsqu'il n'existe plus de préfixe légal pour générer des motifs pouvant vérifier la contrainte.

Prise en compte de contraintes d'agrégats :

L'introduction de certaines contraintes d'agrégats (e.g., moyenne) dans un processus pose plus de difficultés que les contraintes précédentes. En effet, celles-ci ne permettent pas de supprimer les motifs au fur et à mesure du processus, même lorsqu'ils ne vérifient pas les contraintes. Les solutions apportées dans [PHW02] cherchent alors à identifier et à supprimer au plus tôt les motifs ne satisfaisant pas les contraintes et qui ne pourront générer d'autres motifs les respectant. Pour cela, Jian Pei et al. proposent d'une part de supprimer des séquences des bases projetées les items ne pouvant pas former d'occurrences de motifs valides, et d'autre part de supprimer les motifs ne pouvant donner lieu à la génération de motifs respectant les contraintes, en étudiant les motifs générés pour un préfixe donné. Pour cela, les auteurs distinguent les items dont la valeur est inférieure à la valeur de la contrainte (*items faibles*) et les items ayant une valeur supérieure (*items forts*). Par exemple, pour une contrainte de moyenne maximum, les items faibles ne seront pas considérés de la même manière lors du processus puisque la présence de ces items dans les motifs générés ne peut que faire chuter leur moyenne. Inversement, la présence d'items forts dans les motifs ne peut qu'augmenter leur moyenne. Les motifs fréquents ne satisfaisant pas la contrainte de moyenne maximum et contenant des items forts seront alors considérés comme 'critiques' et l'algorithme les traitera avec une attention particulière. Le lecteur pourra se référer à [PHW02] pour une description détaillée du processus.

2.3.3 Approche par listes d'occurrences

Enfin, la troisième famille se compose d'algorithmes [Zak01, Zak98, Zak00, AFGY02, MT96] qui conservent les informations de la base en mémoire. Cette représentation permet de ne pas avoir à effectuer une passe sur la base de séquences lors de l'étape de comptage des supports. Ces algorithmes sont dits basés sur une approche par *listes d'occurrences*. Celles-ci contiennent la description des occurrences des motifs dans les séquences de la base. Ainsi, l'algorithme SPAM [AFGY02] travaille sur des listes d'occurrences dans lesquelles les occurrences d'un motif sont représentées par des valeurs booléennes (présence/absence d'un motif pour une occurrence donnée). L'algorithme MINEPI [MT96], contrairement aux autres algorithmes de cette famille, effectue des extractions de motifs séquentiels sous contraintes à partir d'une seule séquence et se base sur une notion d'occurrences minimales d'un motif afin d'évaluer son support.

Dans cette section, nous donnons un algorithme abstrait (algorithm 3) représentant le principe général des algorithmes de cette famille, en excluant le cas particulier de l'algorithme MINEPI ([MT96]). Cet algorithme est présenté dans sa version explorant l'espace des motifs candidats en largeur (i.e., par niveau). En ce sens, l'ordre de parcours est similaire à celui utilisé par GSP et présenté par la figure 2.2. Cependant, il peut aussi appliquer un parcours en profondeur, ce

qui permet, suivant les jeux de données, d'obtenir de meilleures performances (cf. [Zak01, Zak98] pour une discussion sur les avantages et les inconvénients de chacun de ces parcours). Un exemple de parcours en profondeur de l'espace des candidats est donné figure 2.5.

Les algorithmes de cette famille effectuent une génération préfixée. Ainsi, la génération d'un nouveau motif z de taille $k+1$ est effectuée à partir de deux motifs z_1 et z_2 de taille k qui partagent le même préfixe. Ces motifs sont appelés des *motifs générateurs* de z (par exemple, les motifs générateurs du motif A $\rightarrow$ B $\rightarrow$ C sont les motifs A $\rightarrow$ B et A $\rightarrow$ C). Cette opération, appelée $fusion$ et notée $fusion(z_1, z_2)$, peut donner lieu à la génération de plusieurs motifs candidats. Nous détaillerons un exemple de cette opération de $jointure$ lors de la présentation de l'algorithme *Spade*.

La liste d'occurrences d'un motif z (notée $IdList(z)$) ainsi généré est construite à partir des informations contenues dans les listes d'occurrences de z_1 et de z_2, notées respectivement $IdList(z_1)$ et $IdList(z_2)$. La construction de $IdList(z)$ est appelée $jointure$ et sera notée $jointure(z_1, z_2)$.

Contrairement aux algorithmes des deux premières familles, le comptage du support d'un motif n'est pas effectué par une passe sur la base. Au lieu de cela, les algorithmes basés sur une approche par listes d'occurrences utilisent l'information stockée dans la liste d'occurrence $IdList(z)$ d'un motif z pour connaître son support.

Le choix de représentation d'une occurrence d'un motif dans sa liste diffère selon les algorithmes. Ce choix a un impact direct sur les opérations de *fusion* des motifs et de *jointure* des listes d'occurrences. Ainsi, les opérations de $jointure$ appliquées par l'algorithme SPAM, dont les listes d'occurrences sont des listes booléennes, sont ramenées à des opérations binaires. Les opérations de $fusion$ et de $jointure$ appliquées par l'algorithme *Spade* sont décrites dans la section suivante.

L'algorithme *Spade*

Nous allons maintenant présenter l'algorithme *Spade* [Zak01, Zak98]. Ce dernier utilise des listes d'occurrences dans lesquelles une occurrence est décrite par un identifiant de séquence (*sid*) et par la date d'apparition de son suffixe (*eid*). Dans [Zak01], Mohammed Zaki démontre que la description ainsi faite des occurrences d'un motif est nécessaire et suffisante pour générer tous les motifs séquentiels.

Exemple 6 *La figure 2.7 représente les listes d'occurrences des motifs A, B et A $\rightarrow$ B sur la base exemple de la figure 2.1. Par exemple, la liste d'occurrences du motif A $\rightarrow$ B permet de décrire toutes les occurrences de ce motif dans la base. Il est ainsi possible de savoir que ce motif apparaît dans toutes les séquences de la base puisque toutes les sid apparaissent dans sa liste d'occurrences. De plus, chaque*

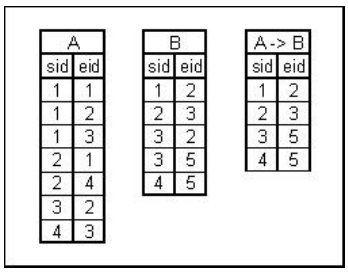
Algorithme 3 (Listes d'occurrences)
Entrée : *Une base de séquences et un support minimum minSup*
Sortie : F, *ensemble de tous les motifs séquentiels fréquents*

Effectue une passe sur la base pour calculer :
- $F_1 = \{i \in I | support(i) \geq minSup\}$
- $IdList(z)$ *pour tous les motifs* z *de* F_1

let $k := 1$
while $F_k \neq \emptyset$ **do**
  let $F_{k+1} := \emptyset$
  for all $z_1 \in F_k$ **do**
    for all $z_2 \in F_k$ **do**
      if z_1 *et* z_2 *ont le même préfixe* **then**
        for all z *obtenu par* $fusion(z_1, z_2)$ **do**
          Construit $IdList(z)$ *par* $jointure(IdList(z_1), IdList(z_2))$.
          Utilise $IdList(z)$ *pour déterminer si* z *est fréquent.*
          if z *est fréquent* **then**
            $F_{k+1} := F_{k+1} \cup \{z\}$
          fi
        od
      fi
    od
  od
  $k := k + 1$
od
output $\bigcup_{1 \leq j < k} F_j$

ligne décrit une occurrence de ce motif. Par exemple, le couple (sid = 1,eid = 2) correspond à une occurrence du motif A → B, dans la première séquence de la base, qui se termine à l'instant 2.

L'opération de jointure correspond à la construction de la liste d'occurrences d'un motif candidat de taille $k + 1$ à partir des listes d'occurrences de ses deux motifs générateurs. Ces *motifs générateurs* correspondent aux motifs fréquents de taille k qui possèdent le même préfixe. Par exemple, les motifs générateurs du motif A → B → C sont les motifs A → B et A → C qui partagent le préfixe commun A.
La génération d'un motif candidat correspond à l'opération de fusion. Celle-ci distingue deux types de motifs : les *motifs séquentiels* et les *motifs événements*. Les *motifs séquentiels* correspondent à des motifs dont le suffixe possède un *eid* strictement supérieur au dernier événement du préfixe, c'est-à-dire qu'une relation



A	
sid	eid
1	1
1	2
1	3
2	1
2	4
3	2
4	3

B	
sid	eid
1	2
2	3
3	2
3	5
4	5

A -> B	
sid	eid
1	2
2	3
3	5
4	5

FIG. 2.7 – Exemple de listes d'occurrences *Spade* pour A,B et A $\rightarrow$B avec la base exemple de la figure 2.1.

temporelle *suivie de* existe entre le préfixe et le suffixe. Inversement, les *motifs événements* sont des motifs pour lesquels le suffixe provient du même événement (c'est-à-dire ayant le même *eid*) que le dernier événement du préfixe.

Opération de Fusion
Soient z_1 et z_2, deux motifs partageant le même préfixe et dont les suffixes respectifs sont notés s_1 et s_2. Le motif résultant de la fusion entre deux motifs dépend de la forme de z_1 et de z_2 (i.e., un motif événement ou un motif séquence). La forme du motif z ainsi généré détermine le type de jointure à appliquer pour calculer la liste d'occurrences de z, $IdList(z)$, à partir des listes d'occurrences $IdList(z_1)$ et $IdList(z_2)$. Ainsi, si le motif z obtenu est un motif événement (respectivement motif séquence), alors l'opération de jointure appliquée sera une jointure équivalente (respectivement jointure temporelle).

Examinons maintenant ces différents cas de génération ainsi que les opérations de jointure associées.

- si z_1 et z_2 sont des *motifs événements* (cas 1) : z_1 et z_2 sont de la forme $z_1 = ps_1$ et $z_2 = ps_2$. Si $s_1 \neq s_2$, le motif obtenu par $fusion(z_1, z_2)$ correspond au motif $z = ps_1s_2$ et sa liste d'occurrences $IdList(z)$ est obtenue par une jointure équivalente : $IdList(z)$=$EqJoin(IdList(z_1),IdList(z_2))$.
- si z_1 est un *motif événement* et z_2 un *motif séquence* (cas 2) : z_1 et z_2 sont alors de la forme $z_1 = ps_1$ et $z_2 = p \rightarrow s_2$. Le motif obtenu par $fusion(z_1, z_2)$ correspond au motif séquence $z = ps_1 \rightarrow s_2$ et sa liste d'occurrences est obtenue par une jointure temporelle :
  $IdList(z)$=$TpJoin(IdList(z_1),IdList(z_2))$.
- si z_1 et z_2 sont des *motifs séquences* : z_1 et z_2 sont de la forme $z_1 = p \rightarrow s_1$ et $z_2 = p \rightarrow s_2$. Si $s_1 \neq s_2$, trois motifs sont générés par $fusion(z_1, z_2)$:
  - (cas 3) le motif événement $z = p \rightarrow s_1s_2$ et sa liste d'occurrences est

obtenue en appliquant une jointure équivalente :
$IdList(z) = EqJoin(IdList(z_1), IdList(z_2))$.
- (cas 4) le motif séquence $z = p \rightarrow s_1 \rightarrow s_2$ et sa liste d'occurrences est obtenue en appliquant une jointure temporelle :
$IdList(z) = TpJoin(IdList(z_1), IdList(z_2))$.
- (cas 5) le motif séquence $z = p \rightarrow s_2 \rightarrow s_1$ et sa liste d'occurrences est obtenue en effectuant une jointure temporelle :
$IdList(z) = TpJoin(IdList(z_2), IdList(z_1))$.

Maintenant, si $s_1 = s_2$, c'est-à-dire que $z_1 = z_2 = p \rightarrow s_1$ (cas 6), alors seul le motif séquentiel $z = p \rightarrow s_1 \rightarrow s_1$ et généré et sa liste d'occurrences est obtenue en effectuant une jointure temporelle : $IdList(z) = TpJoin(IdList(z_1), IdList(z_2))$.

La génération des motifs de taille 2 s'effectue en considérant les items comme des motifs séquentiels dont le préfixe correspond à la séquence $\emptyset$. Les cas de générations qui s'appliquent sont alors les cas 3, 4 et 5 si $s_1 \neq s_2$ et le cas 6 si $s_1 = s_2$.

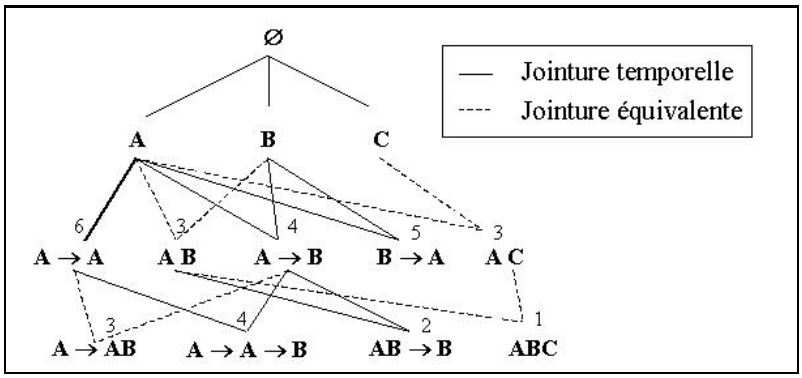


FIG. 2.8 – Illustration des différents cas de génération avec les types de jointures associés.

Exemple 7 *La figure 2.8 représente un graphe partiel des motifs générés à partir d'un alphabet de taille 3. La présence d'un arc en trait plein (resp. pointillés) entre deux motifs signifie que la jointure appliquée est une jointure temporelle (resp. équivalente). Le numéro associé à un motif correspond au cas de génération concerné. Ainsi, la génération du motif $AB \rightarrow B$ correspond au deuxième cas de génération et sa liste d'occurrences est calculée en effectuant une jointure temporelle entre les listes d'occurrences des motifs AB et $A \rightarrow B$.*

Opérations de Jointure

Avant de préciser le principe de ces différentes jointures, précisons le point suivant :
soit z un motif obtenu après opération de fusion entre deux motifs générateurs z_1 et z_2. Les définitions du préfixe et du suffixe d'un motif ainsi que le principe de génération de z permettent de déduire que le suffixe de z_1 appartient au dernier événement du préfixe de z et que le suffixe de z_2 correspond au suffixe de z. Ainsi, la relation temporelle qui existe entre le dernier événement du préfixe de z et son suffixe peut se déduire du lien qu'entretiennent le suffixe de z_1 et le suffixe de z_2.

Les informations concernant les occurrences de ces suffixes se trouvent dans les listes d'occurrences respectives de z_1 et de z_2.

La jointure temporelle permet de vérifier, pour une occurrence donnée, que le suffixe de z va précéder le préfixe de z, c'est-à-dire que le suffixe de z_1 va précéder le suffixe de z_2. De même, la jointure équivalente vérifie, pour une occurrence donnée, que le préfixe et le suffixe de z appartiennent au même événement.

Les deux opérations de jointure procèdent de la façon suivante :

- **jointure temporelle** : ($IdList(z) = TpJoin(IdList(z_1), IdList(z_2))$
  Pour chaque occurrence $\langle sid_1, e_1 \rangle$ provenant de $IdList(z_1)$ et pour chaque occurrence $\langle sid_2, e_2 \rangle$ provenant de $IdList(z_2)$, $TpJoin$ vérifie que $\langle sid_1, e_1 \rangle$ représente une occurrence y_1 qui précède l'occurrence y_2 représentée par $\langle sid_2, e_2 \rangle$. Celui se traduit par $sid_1 = sid_2$ et $e_1 < e_2$, et signifie que les événements de y_1 et de y_2 forment une occurrence z. Le couple $\langle sid_1, e_2 \rangle$ est alors ajouté à $IdList(z)$.
- **jointure équivalente :** ($IdList(z) = EqJoin(IdList(z_1), IdList(z_2))$:
  Pour chaque couple $\langle sid_1, e_1 \rangle$ représentant une occurrence de z_1 et $\langle sid_2, e_2 \rangle$ représentant une occurrence de z_2, la jointure équivalente vérifie que le couple $\langle sid_1, e_1 \rangle$ représente une occurrence y_1 qui se termine à la même date que l'occurrence y_2 représentée par le couple $\langle sid_2, e_2 \rangle$ (i.e., $sid_1 = sid_2$ et $e_1 = e_2$). Si tel est le cas, les occurrences y_1 et y_2 forment une occurrence de z et le couple $\langle sid_1, e_1 \rangle$ est alors ajouté à $IdList(z)$.

Exemple 8 *La figure 2.9 illustre le mécanisme de jointure temporelle entre les listes d'occurrences items A et B sur la base exemple (figure 2.1), pour former la liste d'occurrences du motif $A \rightarrow B$. Seuls les couples d'occurrences <(1,1),(1,2)>, <(2,1),(2,3)>, <(3,2),(3,5)> et <(4,3),(4,5)> donnent lieu à de nouvelles occurrences du motif $A \rightarrow B$. La figure 2.10 illustre le mécanisme de jointure équivalente entre les listes d'occurrences items A et B sur la base exemple (figure 2.1) pour former la liste d'occurrences du motif AB. Seuls les couples d'occurrences <(1,2),(1,2)>, <(3,2),(3,2)> donnent lieu à la création de nouvelles occurrences du motif AB.*

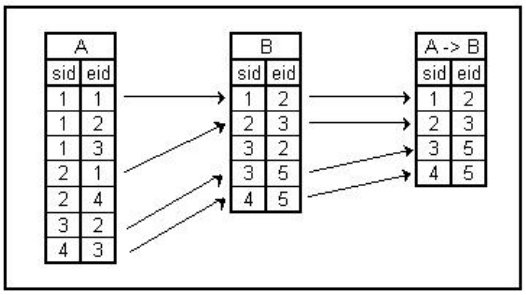


FIG. 2.9 – Exemple d'une jointure temporelle entre les listes $IdList$(A) et $IdList$(B) pour former la liste d'occurrences du motif A $\rightarrow$ B sur la base exemple de la figure 2.1.

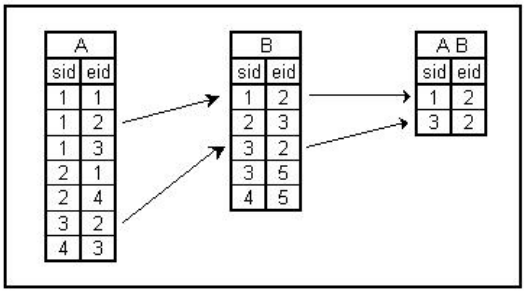


FIG. 2.10 – Exemple d'une jointure équivalente entre les listes $IdList$(A) et $IdList$(B) pour former la liste d'occurrences du motif AB sur la base exemple de la figure 2.1.

Contraintes

Afin de prendre en compte d'autres contraintes, Mohammed Zaki a proposé une évolution de *Spade* dans [Zak00]. Cet algorithme permet d'intégrer des contraintes de temps (fenêtre de temps maximum, gap minimum et gap maximum) ainsi que des contraintes syntaxiques (longueur, largeur, inclusion, exclusion).

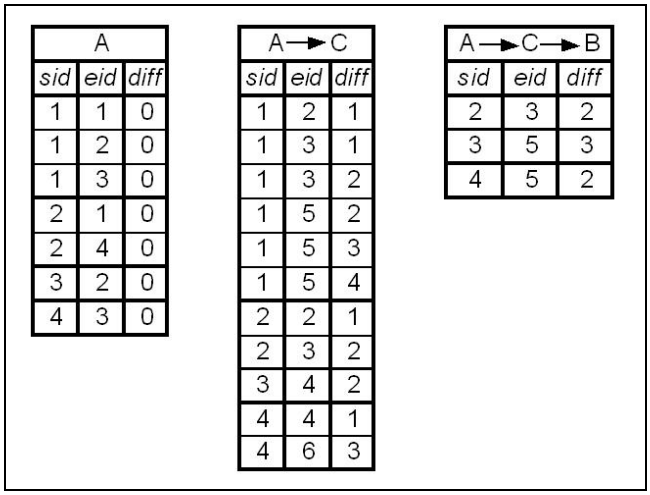
Durant un processus d'extraction, les contraintes syntaxiques interviennent lors de la génération des candidats. Par exemple, si une contrainte de longueur est fixée à $longMax$, alors le processus s'arrêtera dès que $k > longMax$. La gestion de ces contraintes ne sera pas détaillée ici. Le lecteur pourra se référer à l'article [Zak00] pour plus de précisions concernant l'introduction de ce type de contraintes.

Les contraintes de temps, quant à elles, interviennent lors du comptage des supports des candidats puisque celles-ci ont un impact sur la validité des occurrences d'un motif et non plus sur le motif lui-même. Ainsi, une contrainte de fenêtre de temps maximum fixée à 2 peut supprimer l'occurrence A(2) $\rightarrow$ B(5), du motif A $\rightarrow$ B, dans la troisième séquence (cf. figure 2.1) : sa fenêtre de temps est de 3 (5-2), donc supérieure à la contrainte fixée. Cependant, ce motif reste fréquent à un seuil de 75% puisque ses occurrences dans les trois autres séquences respectent cette contrainte.

Listes d'occurrences contraintes :
L'introduction d'une contrainte de fenêtre de temps maximum nécessite de considérer une donnée supplémentaire dans la description d'une occurrence. En effet, pour connaître la durée d'une occurrence d'un motif, il est nécessaire de connaître à la fois la date de début et la date de fin de cette occurrence, ce qui n'est pas le cas avec les occurrences telles qu'elles étaient décrites dans la version non contrainte. Ainsi, les nouvelles listes d'occurrences comportent un champs supplémentaire correspondant à la durée du motif. La figure 2.11 illustre ces nouvelles listes d'occurrences pour A, A $\rightarrow$ C et A $\rightarrow$ C $\rightarrow$ B. Le champs *diff* pour les listes d'occurrences d'un item est nul.

Les opérations de jointure sous contraintes :
Le principe général des opérations de jointure reste similaire au principe présenté dans la section précédente. Cependant, la jointure temporelle doit intégrer les contraintes de temps, puisque celles-ci interviennent au niveau des occurrences d'un motif. Soient $\langle sid_1, e_1, diff_1 \rangle$ le triplet représentant une occurrence de z_1 et $\langle sid_2, e_2, diff_2 \rangle$ le couple représentant une occurrence de z_2, et soit z le motif généré à partir de z_1 et z_2.

La jointure temporelle doit vérifier les points suivants : d'une part, elle doit s'assurer que les occurrences proviennent bien de la même séquence ($sid_1 = sid_2$) et que le dernier événement de z_1 précède le dernier événement de z_2 ($e_1 < e_2$). Ces premiers points sont similaires à ceux appliqués par les jointures non contraintes. Enfin, la jointure temporelle doit vérifier que ces occurrences res-



A		
sid	*eid*	*diff*
1	1	0
1	2	0
1	3	0
2	1	0
2	4	0
3	2	0
4	3	0

A→C		
sid	*eid*	*diff*
1	2	1
1	3	1
1	3	2
1	5	2
1	5	3
1	5	4
2	2	1
2	3	2
3	4	2
4	4	1
4	6	3

A→C→B		
sid	*eid*	*diff*
2	3	2
3	5	3
4	5	2

FIG. 2.11 – Exemple de listes d'occurrences contraintes pour A, A $\rightarrow$ C et A $\rightarrow$ C $\rightarrow$ B avec la base exemple de la figure 2.1.

pectent les contraintes de temps. Pour cela, l'opération effectue les tests suivants : $gapMin \leq eid_2 - eid_1 \leq gapMax$ pour les contraintes de gap minimum et les contraintes de gap maximum, et $diff_1 + (eid_2 - eid_1) \leq winMax$ pour la contrainte de fenêtre de temps maximum. Si tous ces points sont vérifiés, cela signifie que les occurrences considérées de z_1 et de z_2 forment une nouvelle occurrence valide du motif z et le triplet $\langle sid_2, eid_2, diff_1 + (eid_2 - eid_1)\rangle$ est ajouté à la liste d'occurrences de z.

La jointure équivalente, quant à elle, procède de façon similaire à la jointure équivalente non contrainte, c'est-à-dire que pour un couple d'occurrences provenant des listes de z_1 et de z_2, la jointure équivalent va s'assurer qu'il existe bien des occurrences de z_1 et de z_2 qui se terminent à la même date ($sid_1 = sid_2$ et $eid_1 = eid_2$). La façon dont la valeur de $diff$ est calculée n'est pas précisée dans l'article concernant *cSpade*([Zak00]). Cependant, il semble raisonnable de conserver la valeur correspondant à une occurrence dont la taille de fenêtre est la plus petite, i.e., $diff = min(diff_1, diff_2)$.

La contrainte de gap maximum :
Cette contrainte agit aussi au niveau des occurrences d'un motif, mais contrairement aux autres contraintes de temps, une occurrence d'un motif qui ne respecterait pas cette contrainte ne sera pas pour autant supprimée de sa liste d'occurrences. En effet, un motif peut être généré ultérieurement en introduisant un nouvel événement entre un préfixe et son suffixe. Ainsi, les occurrences non va-

lides peuvent le devenir car l'insertion de ce nouvel événement peut réduire le gap qui existait entre son suffixe et le dernier événement de son préfixe. Avant de présenter la solution utilisée par *cSpade* pour pallier à ce problème, illustrons le par l'exemple suivant :

Exemple 9 *Considérons les occurrences des motifs* $B \rightarrow D$ *et* $B \rightarrow F$ *dans la première séquence de la base exemple (figure 2.1), pour illustrer le fait qu'une occurrence non valide peut générer une occurrence valide. On constate que l'occurrence* $B(2) \rightarrow F(5)$ *ne vérifie pas la contrainte de gap maximum si celle-ci est fixée à 2 (*$eid(F) - eid(B) = 5 - 2 = 3 > gapMax$*). Considérons maintenant que ces motifs soient utilisés pour générer le motif* $B \rightarrow D \rightarrow F$*. On constate que l'occurrence de ce motif dans la première séquence,* $B(2) \rightarrow D(4) \rightarrow F(5)$*, respecte la contrainte entre chacun de ses événements (*$eid(F) - eid(D) = 5 - 4 = 1 < gapMax$ *et* $eid(D) - eid(B) = 4 - 2 = 2 = gapMax$*) alors qu'elle a été générée à partir d'une occurrence non valide.*

La génération des candidats en présence d'une contrainte de gap maximum :
Le processus de génération utilisé dans *Spade* n'est plus adapté lorsqu'une contrainte de gap maximum est intégrée au processus d'extraction. En effet, pour les raisons précisées ci-dessus, la contrainte de gap maximum ne permettrait plus de filtrer les candidats ce qui lui retirerait un intérêt non négligeable. Afin de pallier à ce problème, l'algorithme *cSpade* propose de revoir le principe de génération des candidats de taille $k+1$ afin de rendre à la contrainte de gap maximum son aspect sélectif. Cette nouvelle génération ne s'effectue plus en considérant deux à deux les motifs fréquents de taille k, mais en ajoutant un item à la fin de tous les motifs appartenant à F_k. Cet item provient d'un motif de F_2, c'est-à-dire d'un motif appartenant à l'ensemble des motifs fréquents de taille 2, dont le préfixe est identique au dernier item du suffixe du k-motif. Ainsi, le 4-motif AB $\rightarrow$ C $\rightarrow$ D ne sera plus généré à partir des 3-motifs AB $\rightarrow$ C et AB $\rightarrow$ D, mais à partir du 3-motif AB $\rightarrow$ C et du 2-motif C $\rightarrow$ D. Ce mode de génération permet ainsi de ne pas avoir à conserver les motifs qui ne respectent pas la contrainte de gap maximum. Le fait de prendre en compte les 2-motifs et non les 1-motifs évite d'avoir à générer trop de candidats 'parasites'. En effet, ce mode de génération engendre un nombre important de candidats 'parasites' ce qui alourdit le processus global. Cependant, les expériences présentées dans [Zak00], montrent que la sélectivité de la contrainte de gap maximum permet de conserver une approche rentable. De plus, ce mode de génération n'est nécessaire qu'en présence d'une contrainte de gap maximum et n'est pas utile lorsque les extractions ne font intervenir qu'une contrainte de gap minimum et/ou de fenêtre de temps maximum.

2.4 Synthèse et conclusion

Dans cet état de l'art, nous avons développé certaines techniques d'extraction de motifs séquentiels dans des bases de séquences. Cette étude a fait apparaître trois familles d'algorithmes basées sur des techniques de génération des motifs candidats et de comptage des supports qui leur sont propres. Le tableau de la figure 2.12 résume ces différentes techniques pour les principaux algorithmes cités dans cet état de l'art et précise pour chacun d'eux les contraintes qu'ils permettent d'intégrer. Ainsi, les algorithmes de la famille *APriori* (e.g., GSP [SA96] et PSP [MCP98]), appliquent un parcours en largeur des motifs candidats et effectuent une passe complète sur la base afin d'évaluer le support des motifs candidats générés. Les algorithmes de la deuxième famille, c'est-à-dire les algorithmes basés sur une approche par *Projection* (e.g., FreeSpan [HHMAC+00] et PrefixSpan [PHMAP01]), effectuent un parcours en profondeur des candidats et procèdent à des projections successives de la base sur les motifs fréquents. Ces bases projetées sont utilisées lors du comptage des différents supports. Enfin, la dernière famille, regroupe les approches par *listes d'occurrences* (e.g., *Spade* [Zak01, Zak98] et *cSpade* [Zak00]). Ces algorithmes conservent les apparitions des motifs dans les séquences en mémoire et s'appuient sur les informations qu'elles contiennent pour compter le support de chaque candidat. Ces algorithmes peuvent effectuer un parcours en profondeur ou en largeur de l'espace des motifs candidats.

		Parcours de candidats		Contraintes			
		Profondeur	Largeur	Fréquence minimum	Contraintes de Temps	Contraintes Syntaxiques	Autres Contraintes
Approche *A Priori*	GSP		■	■	■	■	
	PSP		■	■	■	■	
	Spirit		■	■		ER	
	RE-Hackle		■	■		ER	
Approche par *Projection*	FreeSpan	■		■			
	PrefixSpan	■		■			
	PrefixSpan+contraintes	■		■	■	■	■
Approche par *Listes d'occurrences*	Spade	■	■	■			
	cSpade	■	■	■	■	■	
	Spam	■		■			

FIG. 2.12 – Synthèse des algorithmes existants (**ER** = Expression Régulière).

Toutes ces approches ont montré leur intérêt dans de nombreux contextes applicatifs. A ce jour, aucune étude n'a démontré que telle ou telle approche était meilleure qu'une autre dans tous les cas. Le tableau de la figure 2.13, permet cependant d'illustrer les principaux points faibles que présentent chaque famille, au niveau de leur consommation mémoire et de leur principaux coûts de calculs.

	Consommation mémoire	Coûts de calculs
Apriori	Stockage des motifs candidats	Comptage des motifs candidats
Projection	Stockage des bases projetées	Projection des bases des motifs fréquents
Listes d'occurrences	Stockage des listes d'occurrences	Jointures entre les listes d'occurrences

FIG. 2.13 – Origine de la consommation mémoire et des coûts de calculs pour chaque famille d'algorithmes.

Ainsi, la principale consommation mémoire des algorithmes de la première famille repose sur le stockage des motifs candidats d'un même niveau (puisque le parcours s'effectue en largeur), et les coûts de calculs proviennent de l'étape de comptage des candidats qui correspond principalement au coût d'identification des motifs dans les séquences de la base.

Pour les algorithmes de la deuxième famille, les principaux coûts de consommation mémoire et de calculs proviennent de la construction et du stockage des bases projetées en mémoire (une base projetée pour un motif fréquent).

Enfin, les approches par listes d'occurrences doivent d'une part stocker les listes en mémoire, et d'autre part effectuer des jointures successives entre ces listes.

La lecture de ce tableau met en évidence le fait que les principaux points faibles de chaque approche proviennent de leur spécificité en matière de génération des motifs candidats et du comptage de leur support. Des efforts ont donc été fait afin d'optimiser ces différents points. Par exemple, les algorithmes de la première famille ont développés des représentations permettant de réduire l'espace mémoire consommé par les motifs candidats et de faciliter leur comptage et les algorithmes de la deuxième famille ont procédés à des projections virtuelles des bases afin de réduire leur coût de stockage.

Les paramètres qui influencent la consommation mémoire et les coûts de calculs sont différents pour chacune des familles. C'est pour cette raison qu'il n'est pas possible de conclure qu'une approche est définitivement meilleure que les deux autres, dans tous les cas (tous types de données et paramètres d'extraction).

Par exemple, si lors d'une extraction il existe un très grand nombre de candidats de même taille, alors les algorithmes de la première famille sont mis en défaut par l'espace mémoire qui leur est nécessaire pour les stocker simultanément. En revanche, les algorithmes des deux autres familles, seront pénalisés dans le cas d'une très grande base (taille des bases projetées et des listes d'occurrences importantes) qui peut les conduire à avoir recours à la mémoire secondaire (disque dur), entraînant ainsi une dégradation rapide de leurs performances. En ce qui concerne

les coûts de calculs, ceux-ci sont influencés de façon locale lors du comptage du support des motifs. Ainsi, plus les listes d'occurrences des motifs sont longues (motifs apparaissant fréquemment dans la base et dans une même séquence), plus les coûts de jointure seront importants.

Ainsi, les deux dernières approches seront plus appropriées aux extractions à faible support et/ou aux extractions de motifs longs. En effet, même si leur consommation mémoire représente leur limite principale, l'économie effectuée lors des évaluations de supports augmente au cours de leur exécution car la taille des bases projetées dans un cas et la taille des listes d'occurrences dans l'autre cas, tendent à se réduire tout au long du processus, diminuant alors fortement les coûts de projection et de jointure.

Dans ce chapitre, nous avons étudié différentes techniques d'extraction de motifs séquentiels fréquents, contraints ou non. Cette étude nous a permis de mettre en avant trois familles d'approches spécifiques, dont les principaux points faibles ont été discutés dans cette section.

Dans le chapitre suivant, nous présentons les choix qui nous ont conduit à utiliser les approches par listes d'occurrences dans notre contexte applicatif. Nous introduisons ensuite les limites que présentent les algorithmes basés sur une approche par listes d'occurrences, dans le cas particulier de données contenant des répétitions consécutives, telles que celles que nous avons eu à traiter. Nous présentons alors une solution qui permet de réduire la taille totale des listes d'occurrences considérées sans perte d'information pour les extractions. La représentation correspondante consiste à représenter plusieurs occurrences consécutives par une seule occurrence, plus générale, qui contient autant d'information.

Nous développons ainsi un algorithme (*GoSpade* [LRBE03b]) d'extraction de motifs séquentiels fréquents dédié à l'utilisation de listes d'occurrences généralisées. Nous montrons d'une part que les résultats qu'il retourne sont justes et complets, et d'autre part qu'il est particulièrement bien adapté à la présence de répétitions consécutives dans les séquences, comparé à l'algorithme *Spade* [Zak01, Zak98].

Nous expliquons ensuite de quelle manière ces nouvelles occurrences peuvent être complétées afin de pouvoir être utilisées dans le cas d'extractions faisant intervenir des contraintes de temps (e.g., contrainte de fenêtre de temps maximum). Enfin, nous proposons un second algorithme (*GoSpec* [LRBE03a]), correspondant à une version contrainte du premier algorithme et utilisant ces occurrences généralisées complétées. Comme pour *GoSpade*, une preuve de complétude et de justesse a été établie et des expérimentations ont permis de montrer qu'il était plus efficace que l'algorithme *cSpade* [Zak00] (version contrainte de *Spade*), lors d'extractions effectuées sur des données contenant des répétitions consécutives.

Chapitre 3

Algorithmes

Un des aspect de notre travail concerne le traitement de données financières et notamment la recherche de motifs séquentiels fréquents dans ces données. L'étude des différents algorithmes existants nous a permis d'identifier la famille d'approches basées sur l'utilisation de listes d'occurrences, comme étant une solution intéressante pour l'extraction de motifs séquentiels. En effet, les principaux algorithmes de cette famille (*Spade* et *cSpade*) présentent de bonnes performances à bas support comparé à la famille APriori, et possèdent l'avantage de pouvoir intégrer des contraintes temporelles, ce qui n'était alors pas le cas des approches par projection.

Ces constats ont motivés notre choix de développer les deux principaux algorithmes de cette famille, c'est-à-dire l'algorithme *Spade* [Zak98, Zak01] et sa version contrainte *cSpade* [Zak00], pour rechercher les motifs séquentiels fréquents dans nos données.

Les premières expérimentations effectuées sur ces jeux consommaient malheureusement beaucoup d'espace mémoire, ce qui avait pour effet de ralentir considérablement les exécutions et parfois même de les rendre impossibles. Cette consommation était due à la taille importante des listes d'occurrences des motifs. Une étude plus précise du contenu de ces listes nous a permis de réaliser que les séquences traitées contenaient beaucoup de répétitions consécutives de certains items. Un gain nous est apparu possible. Celui-ci consiste à condenser les informations contenues dans les listes en représentant plusieurs occurrences consécutives en une occurrence plus générale qui contient la même information. Cette représentation permet de réduire d'une part l'espace mémoire nécessaire à une extraction de motifs séquentiels, et d'autre part le temps d'exécution total d'un processus.

Dans ce chapitre nous expliquons l'impact de la présence de répétitions consécutives dans les données sur le contenu des listes d'occurrences classiques telles que celles utilisées par *Spade* [Zak01, Zak98], et donnons le principe de compactage des listes dans le cas d'extraction de motifs séquentiels fréquents, sans considération

d'autres contraintes. Nous présentons ensuite l'algorithme *GoSpade* (section 3.2) proposé pour effectuer des extractions en utilisant ce nouveau type d'occurrences, suivi d'une version permettant de considérer des contraintes temporelles (fenêtre de temps maximum, gap minimum et maximum), en plus de la contrainte de fréquence minimum (algorithme *GoSpec*, section 3.3).

3.1 Motivations

Considérons la base de la figure 3.1, composée de deux séquences construites à partir d'un alphabet composé des items A et B. Considérons aussi la liste d'occurrences de *Spade* pour l'item A sur cette base (première table de la figure 3.2). On peut remarquer que cet item apparaît de façon consécutive dans les séquences, c'est-à-dire qu'il apparaît dans tous les événements entre les *eids* 1 et 3 dans la première séquence et entre les *eids* 1 et 4 dans la deuxième séquence. Une telle situation peut apparaître dans plusieurs types de bases de données et en particulier lorsque les séquences sont construites à partir de données quantitatives telles que des séries temporelles présentant de faibles variations. Les listes d'occurrences comme celles utilisées dans *Spade* ([Zak01, Zak98]) contiennent une ligne par occurrence, c'est-à-dire 3 lignes pour représenter les occurrences de A dans la première séquence et 4 lignes pour celles de la seconde.

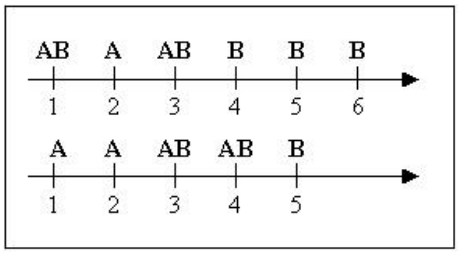


FIG. 3.1 – Exemple base contenant des répétitions consécutives.

La présence de répétitions dans les séquences multiplie le nombre d'occurrences d'un motif et augmente d'autant la taille de sa liste d'occurrences, sans pour autant participer à sa fréquence. Considérons le cas très simple d'une séquence composée des événements {A}{B} et d'une séquence contenant des répétitions sur l'item B, {A}{B}{B}{B}. La gestion de la première séquence par l'algorithme *Spade* pour la liste du motif A $\rightarrow$ B nécessite une seule occurrence, alors que la second en nécessite 3 (une par date de fin de B).

L'augmentation de ce nombre d'occurrences a pour effet d'alourdir le coût de jointure total. Considérons par exemple la figure 3.3. Cette figure représente les

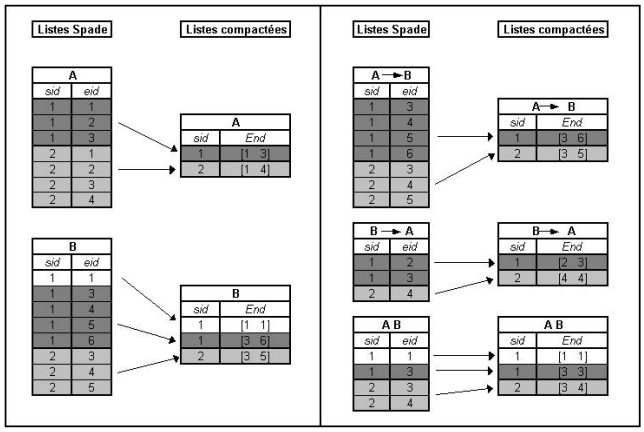


FIG. 3.2 – Listes d'occurrences *Spade* vs. Listes d'occurrences compactées.

résultats d'extractions effectuées sur deux jeux de données synthétiques ($set1\_r0$ et $set1\_r5$). Le jeu $set1\_r5$ contient les mêmes séquences que le jeu $set1\_r0$ dans lesquelles un certain nombre de répétitions consécutives a été ajouté pour quelques items (la description de ces jeux et de leur construction sera détaillée dans la section 3.2.4). Les deux courbes de cette figure représentent le coût, en terme de temps d'exécution global, nécessaire à l'extraction d'un certain nombre de motifs séquentiels fréquents sur chacun des jeux (i.e. pour différents seuils de fréquence minimum). Ces courbes montrent que le temps d'exécution nécessaire à *Spade*, pour extraire un nombre donné de motifs séquentiels fréquents, est plus important sur le jeu contenant des répétitions consécutives ($set1\_r5$).

Nous introduisons le concept d'*occurrence généralisée* afin de compacter les occurrences consécutives d'un motif. Par exemple, figure 3.2, les 3 occurrences consécutives de l'item A dans la première séquence peuvent être représentées par une seule occurrence généralisée de la forme $\langle 1, [1, 3]\rangle$. Cette occurrence généralisée contient l'identifiant de la séquence (*sid*, ici égale à 1) et un intervalle [1,3] qui contient les *eids* de toutes les occurrences consécutives ($\{1,2,3\} \in [1,3]$). Lorsqu'un motif est composé de plusieurs événements, l'intervalle contient les *eids* des occurrences consécutives du dernier événement. Par exemple, les 4 occurrences du motif A $\rightarrow$ B dans la première séquence de la figure 3.2 sont représentées par une seule occurrence généralisée $\langle 1, [3, 6]\rangle$.

L'utilisation d'une telle représentation permet de réduire de manière significative la taille des listes d'occurrences, dès lors que des occurrences consécutives apparaissent dans la base de séquences. Cette représentation compacte des listes d'occurrences s'appelle *liste d'occurrences généralisées* et sera notée *GoIdList*.

La figure 3.2 illustre, pour les items A et B ainsi que pour les motifs A $\rightarrow$

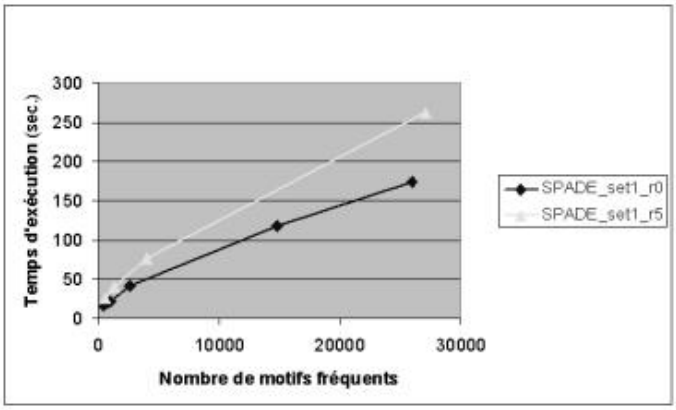


FIG. 3.3 – Evolution du temps d'exécution global de *Spade* sur des jeux contenant des répétitions consécutives.

B, B $\rightarrow$ A et AB, les réductions obtenues lorsque les listes sont ramenées à des listes d'occurrences généralisées. Par exemple, la liste d'occurrences de l'item A, $IdList$(A), contient 7 occurrences alors que sa liste d'occurrences généralisées, $GoIdList$(A), ne contient que 2 occurrences généralisées. De façon plus générale, la figure 3.4 montre l'évolution de la longueur moyenne des listes pour des extractions effectuées sur les jeux synthétiques $set1\_r0$ et $set1\_r5$, lorsque celles-ci sont effectuées en utilisant des listes d'occurrences de *Spade* et des listes d'occurrences généralisées. Les courbes montrent que, pour un nombre de motifs fréquents extraits, les longueurs moyennes des listes utilisées par *Spade* sont plus importantes que celles utilisées par les listes d'occurrences généralisées lorsque le jeu contient des répétitions consécutives ($set1\_r5$). De plus, les longueurs moyennes des listes utilisées sont similaires lorsque les extractions portent sur le jeu $set1\_r0$, c'est-à-dire sur le jeu ne comprenant pas de répétitions consécutives ajoutées (la légère différence provenant des répétitions inhérentes à sa construction).

Comme pour les listes d'occurrences utilisées par *Spade*, les listes d'occurrences généralisées des motifs sont construites à partir des listes de deux autres motifs. La figure 3.2 montre, avec les motifs A $\rightarrow$ B, B $\rightarrow$ A et AB, générés à partir des motifs A et B, que la réduction des listes se propage aussi durant le processus de *jointure*.

Dans la section suivante (section 3.2), nous présentons l'algorithme *GoSpade*, algorithme permettant d'effectuer des extractions de motifs séquentiels basées sur l'utilisation de listes d'occurrences généralisées. Cet algorithme agit en particulier sur les opérations de jointure, $TpJoin$ et $EqJoin$(section 2.3.3), qui doivent être spécifiques à la gestion de cette nouvelle structure des occurrences.

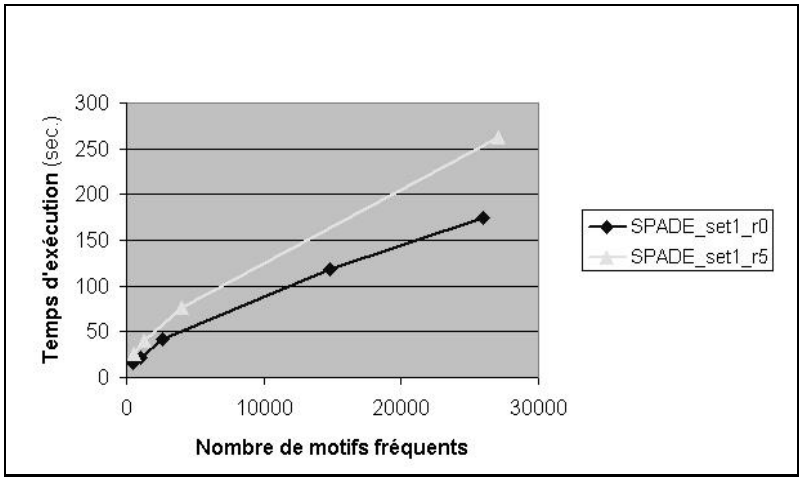


FIG. 3.4 – Longueur moyenne des listes d'occurrences de *Spade* vs. longueur moyenne listes d'occurrences généralisées de *GoSpade*.

3.2 L'algorithme *GoSpade*

L'algorithme *GoSpade* [LRBE03b] est un algorithme d'extraction de motifs séquentiels fréquents particulièrement bien adapté pour le traitement de bases de séquences contenant des répétitions consécutives. Celui-ci appartient à la dernière famille d'algorithmes, basés sur une approche par listes d'occurrences (section 2.3.3), et présente la particularité de travailler à partir d'occurrences généralisées qui permettent de compacter les occurrences utilisées par *Spade*.

Dans la section suivante nous présentons la structure des listes d'occurrences généralisées. Nous introduisons ensuite l'algorithme général de *GoSpade* et établissons la preuve de sa correction. Enfin, nous terminerons la présentation de *GoSpade* avec un ensemble de résultats expérimentaux, effectués sur des jeux de données synthétiques, qui permettent de montrer l'intérêt de *GoSpade* par rapport à *Spade*.

3.2.1 *GoIdLists* : des listes d'occurrences généralisées

Une *occurrence généralisée* est une représentation compacte de plusieurs occurrences d'un motif z et contient les informations suivantes :

- Un identifiant *sid* correspondant à l'identifiant d'une séquence dans laquelle le motif z apparaît.
- Un intervalle $[min,max]$ représentant les occurrences consécutives du dernier

événement d'un motif z.
Une occurrence généralisée est représentée par un tuple $\langle sid, [min, max]\rangle$.

Une liste d'occurrences généralisées, notée $GoIdList$, est une liste contenant toutes les occurrences généralisées d'un motif séquentiel. La liste d'occurrences généralisées d'un motif z sera notée $GoIdList(z)$.

3.2.2 L'algorithme *GoSpade*

Le principe général de *GoSpade* est similaire à celui des algorithmes de la dernière famille présentée dans la section 2.3.3 et le processus de génération des candidats reste le même que celui de *Spade* (i.e., un nouveau motif z est généré à partir de deux motifs générateurs z_1 et z_2 partageant le même préfixe p, cf. section-spade).

La différence repose sur la nature des éléments qui composent les listes d'occurrences, les listes utilisées dans *GoSpade* étant constituées d'occurrences généralisées. Ainsi, les opérations de jointure, $TpJoin$ et $EqJoin$, doivent être adaptées à cette nouvelle structure de listes. Les algorithmes correspondants sont présentés dans la partie 3.2.2, toutefois, dans un soucis de clarté, nous commençons par redonner l'algorithme général d'extraction utilisé par *GoSpade* avec l'algorithme 4. Cet algorithme est similaire à l'algorithme (algorithme 3) présenté dans la section 2.3.3 mais abordé cette fois-ci avec un parcours en profondeur. De plus, nous présentons sous forme algorithmique les opérations de $fusion$ (algorithme 5) et de $jointure$ (algorithme 6) décrites de façon informelle dans la section concernant l'algorithme *Spade* (section 2.3.3).

L'algorithme général *GoSpade*

Cet algorithme effectue un parcours en profondeur de l'espace des séquences candidates possibles. Ainsi, tous les 1-motifs fréquents donnent lieu à la construction d'un sous-arbre traité indépendamment.
Précisons que les listes d'occurrences généralisées calculées sont conservées pour toute la durée d'exécution. De plus, les valeurs de $F_{[\alpha]}$ sont stockées de façon globale pour toute la durée de l'exécution et sont initialisées à vide avant le premier appel de *GoSpade*. On pourra se référer à la figure 2.5 (page 41) pour une illustration d'un parcours en profondeur de l'espace des candidats. Ainsi, à partir de deux motifs générateurs z_1 et z_2, *GoSpade* va générer plusieurs motifs candidats (opération de $fusion$ présentée avec l'algorithme 5) et construire pour chacun d'eux leur nouvelle liste d'occurrences généralisées qui permettront d'évaluer leur support. Ces listes sont construites en appliquant une opération de $jointure$

Algorithme 4 (*GoSpade*)
Entrée : *Une base de séquences β, un support minimum minSup et un motif p représentant le préfixe courant.*
Sortie : *F, ensemble de tous les motifs séquentiels fréquents dans β.*
Premier appel : *GoSpade(β,minSup,$\emptyset$)*

1. **if** *($p = \emptyset$)* **then**
2. *Construire GoIdList(i) pour tout $i \in I$*
3. *$F_{[\emptyset]} := \{i \in I | support(i) \geq minSup$ dans $\beta\}$*
4. *$Freq := F_{[\emptyset]}$*
5. **else**
6. *$Freq := \emptyset$*
7. **fi**
8. **for all** *$z_i \in F_{[p]}$* **do**
9. **for all** *$z_j \in F_{[p]}$, tel que $z_j \geq z_i$* **do**
10. **for all** *z obtenu par $fusion(z_i, z_j)$* **do**
11. *GoIdList(z) := jointure(GoIdList(z_i),GoIdList(z_j),z)*
12. *Utilise GoIdList(z) pour déterminer si z est fréquent*
13. **if** *z est fréquent* **then**
14. *$F_{[prefixe(z)]} := F_{[prefixe(z)]} \cup \{z\}$*
15. *$Freq := Freq \cup \{z\}$*
16. **fi**
17. **od**
18. **od**
19. *Freq := Freq$\cup$ GoSpade(β,minSup,z_i)*
*20.***od**
*21.***output** *Freq*

FIG. 3.5 – Algorithme général (*GoSpade*).

adéquate (celle-ci est identifiée après avoir appliqué l'algorithme général de jointure présenté dans l'algorithme 6).

Les opérations de fusion et de jointure

Ces algorithmes reprennent sous forme algorithmique les opérations de $fusion$ et de $jointure$ décrites dans la section concernant l'algorithme *Spade* (section 2.3.3). L'algorithme représentant l'opération de $fusion$ (algorithme 5) permet de générer un ensemble de motifs séquentiels candidats à partir de deux motifs générateurs. Cet algorithme reprend les différents cas de générations présentés dans la section 2.3.3.

Nous présentons maintenant les opérations de $jointure$ spécifiques adaptées au traitement de listes d'occurrences généralisées, appelées $TpJoin$ (algorithme 7) et $EqJoin$ (algorithme 9).

Les opérations de jointure de l'algorithme *GoSpade*

Les algorithmes $TpJoin$ (algorithme 7) et $EqJoin$ (algorithme 9) génèrent une nouvelle liste d'occurrences généralisées, $GoIdList$, à partir des listes de deux motifs générateurs z_1 et z_2. Ces deux algorithmes procèdent de façon similaire. Les deux boucles imbriquées permettent de considérer toutes les paires d'occurrences généralisées composées d'une occurrence provenant de $GoIdList(z_1)$ et d'une occurrence provenant de $GoIdList(z_2)$. Pour chaque paire ($\langle sid_1, [min_1, max_1]\rangle$, $\langle sid_2, [min_2, max_2]\rangle$), ainsi constituée, ces algorithmes font appel à une fonction permettant de construire une nouvelle occurrence généralisée : respectivement $LocalTpJoin$ (algorithme 8) et $LocalEqJoin$ (algorithme 10).

L'algorithme 7 commence par vérifier que $min_1 < max_2$ ce qui signifie qu'il existe au moins une occurrence contenue dans $\langle sid_1, [min_1, max_1]\rangle$ qui se termine avant la fin d'au moins une occurrence contenue dans $\langle sid_2, [min_2, max_2]\rangle$.

Exemple 10 *Illustrons le principe général de ces algorithmes, à partir de notre base exemple (figure 3.1) et des listes de la figure 3.2, et considérons l'appel à TpJoin(IdList(B),IdList(A)) qui a pour objectif de construire la liste d'occurrences généralisées pour le motif* $B \rightarrow A$.

La première paire d'occurrences considérée correspond à la paire ($\langle 1, [1, 1]\rangle$,$\langle 1, [1, 3]\rangle$*), composée par la première occurrence généralisée de IdList(B) (*$\langle 1, [1, 1]\rangle$*) et la première occurrence généralisée de IdList(A) (*$\langle 1, [1, 3]\rangle$*). Le traitement de cette première paire d'occurrences permet de générer une nouvelle occurrence généralisée,* $\langle 1, [2, 3]\rangle$*, pour le motif* $B \rightarrow A$*. Le test de la ligne 3 de l'algorithme*

Algorithme 5 ($fusion$)
Entrée : *Deux motifs z_1, z_2 partageant le même préfixe p.*
Sortie : *Ensemble des motifs générés à partir de z_1 et z_2*

1. **if** $z_1 = ps_1$ *and* $z_2 = ps_2$ **then**
2. **if** $s_1 \neq s_2$ **then**
3. $z := ps_1s_2$ *//Cas de génération 1*
4. $Z := Z \cup \{z\}$
5. **fi**
6. **elsif** $z_1 = ps_1$ *and* $z_2 = p \rightarrow s_2$ **then**
7. $z := ps_1 \rightarrow s_2$ *//Cas de génération 2*
8. $Z := Z \cup \{z\}$
9. **elsif** $s_1 \neq s_2$ **then**
10. $z := p \rightarrow s_1s_2$ *//Cas de génération 3*
11. $Z := Z \cup \{z\}$
12. $z := p \rightarrow s_1 \rightarrow s_2$ *//Cas de génération 4*
13. $Z := Z \cup \{z\}$
14. $z := p \rightarrow s_2 \rightarrow s_1$ *//Cas de génération 5*
15. $Z := Z \cup \{z\}$
16. **else**
17. $z := p \rightarrow s_1 \rightarrow s_2$ *//Cas de génération 6*
18. $Z := Z \cup \{z\}$
19. **fi**
20. **fi**
21. **fi**
*22.***output** Z

FIG. 3.6 – Algorithme de fusion.

TpJoin (algorithme 7), permet de vérifier qu'il existe bien une occurrence de A parmi celles représentées par $\langle 1, [1, 3]\rangle$ *qui succède à au moins une occurrence de B représentée par* $\langle 1, [1, 1]\rangle$. *Pour cela, il vérifie que l'occurrence au plus tôt de B, ici l'occurrence unique B(1), précède au moins l'occurrence au plus tard de A, ici l'occurrence A(3). Ainsi, l'appel à LocalTpJoin est assuré de retourner au moins une occurrence du motif $B \rightarrow A$, soit l'occurrence $B(1) \rightarrow A(3)$. Maintenant, l'appel à LocalTpJoin (algorithme 8) va générer cette nouvelle occurrence généralisée. Pour cela, la ligne 4 va permettre de rechercher l'occurrence au plus tôt de A, parmi les occurrences représentées par* $\langle 1, [1, 3]\rangle$, *qui succède à l'occurrence au plus tôt de B, ici B(1). Celle-ci correspond à l'occurrence A(2), d'où $min = 2$. La première occurrence de A, l'occurrence A(1), ne permet pas de*

Algorithme 6 (*jointure*)
Entrée : *Deux listes d'occurrences généralisées GoIdList(z_1), GoIdList(z_2) et le motif z généré à partir de z_1 et de z_2*
Fonctions appelées : *TpJoin et EqJoin*
Sortie : *La liste d'occurrences généralisées de z*

1. **if** *z est un motif événement* **then**
2. *GoIdList(z) := EqJoin(GoIdList(z_1), GoIdList(z_2))*
3. **elsif** *($suffixe(z) = suffixe(z_1)$ and $suffixe(z_1) \neq suffixe(z_2)$)* **then**
4. *GoIdList(z) := TpJoin(GoIdList(z_2), GoIdList(z_1))*
5. **else**
6. *GoIdList(z) := TpJoin(GoIdList(z_1), GoIdList(z_2))*
7. **fi**
8. **fi**
9. **output** *GoIdList(z)*

FIG. 3.7 – Algorithme de jointure.

générer une occurrence du motif puisque l'occurrence de B, B(1), apparaît à la même date que A(1).

Le traitement de la paire d'occurrences (⟨1, [3, 6]⟩,⟨1, [1, 3]⟩) ne donnera pas lieu à la construction d'une occurrence généralisée pour le motif $B \rightarrow A$. En effet, ces occurrences ne vérifient pas le premier test (ligne 3) de TpJoin (algorithme 7) qui permet de s'assurer qu'il existe au moins une occurrence de B qui précède au moins une occurrence de A.

L'algorithme de jointure *TpJoin* (resp. *EqJoin*) fait appel à *LocalTpJoin* (resp. *LocalEqJoin*) puis teste ligne 5 (resp. ligne 4) que l'occurrence généralisée retournée est valide. Si tel est le cas, alors cette nouvelle occurrence généralisée peut être ajoutée à la liste d'occurrences généralisées en cours de construction (ligne 6, resp. ligne 5). Ces algorithmes se terminent après avoir considéré tous les couples d'occurrences généralisées ($\langle sid_1, [min_1, max_1]\rangle$, $\langle sid_2, [min_2, max_2]\rangle$) et retournent une nouvelle liste, *GoIdList*.

L'algorithme 8, *LocalTpJoin*, génère une nouvelle occurrence généralisée à partir de deux occurrences généralisées données en entrée de la procédure. Il commence par vérifier que les deux occurrences fournies proviennent de la même séquence, c'est-à-dire que $sid_1 = sid_2$ (ligne 3). Les lignes 4 à 8 génèrent cette nouvelle occurrence. La ligne 4 fixe la valeur *min* de la nouvelle occurrence généralisée avec le plus petit élément de $[min_2, max_2]$ qui est strictement supérieur à min_1. Cela signifie que *min* est la première occurrence contenue dans $\langle sid_2, [min_2, max_2]\rangle$ qui succède strictement à la première occurrence de $\langle sid_1, [min_1, max_1]\rangle$.

Algorithme 7 (*TpJoin*)
Entrée : *GoIdList*(z_1), *GoIdList*(z_2)
Fonction appelée : *LocalTpJoin (algorithme 8)*
Sortie : *GoIdList, une nouvelle liste d'occurrences généralisées*

Initialise GoIdList à la liste vide.
1. **for all** $occ_1 \in GoIdList(z_1)$ **do**
2. **for all** $occ_2 \in GoIdList(z_2)$ **do**
3. **if** $(min_1 < max_2)$ **then**
4. **let** $\langle v, add \rangle$:= *LocalTpJoin*
$(\langle sid_1, [min_1, max_1] \rangle, \langle sid_2, [min_2, max_2] \rangle)$
5. **if** *add* **then**
6. *Ajoute v à GoIdList*
7. **fi**
8. **fi**
9. **od**
10. **od**
11. **output** *GoIdList*

FIG. 3.8 – Algorithme de jointure temporelle (*TpJoin*).

La ligne 5 fixe la valeur de *sid* et enfin, la ligne 6 fixe la valeur *max* de la nouvelle occurrence généralisée à la valeur de max_2 (la position de la dernière occurrence de z_2).

L'algorithme 10, *LocalEqJoin*, commence par s'assurer que les deux occurrences généralisées proviennent de la même séquence et effectue (ligne 4) l'intersection des deux intervalles $[min_1, max_1]$ et $[min_2, max_2]$. Si cette intersection n'est pas vide, cela signifie qu'il existe des occurrences du nouveau motif qui se terminent à chaque *eid* de cette intersection. Enfin, l'algorithme attribue le résultat de cette intersection à [*min*,*max*] et fixe la valeur de *sid*.

Dans la section suivante, nous montrons que les listes d'occurrences généralisées sont justes et complètes, ce qui nous permet d'affirmer que les comptages des supports à partir de ces listes sont correctes.

3.2.3 Correction de l'algorithme *GoSpade*

Soit la définition préliminaire suivante :

Définition 10 ***(v représente y)*** Soit y une occurrence d'un motif z dans une séquence S provenant d'une base de séquences β. Soit $GoIdList(z)$ la liste d'occurrences généralisées de ce motif et v une occurrence généralisée de $GoIdList(z)$, représentée par le tuple $\langle sid, [min, max] \rangle$. On dit que v *représente* y si $sid = Id(S)$

Algorithme 8 (*LocalTpJoin*)
Entrée : *Deux occurrences généralisées* $\langle sid_1, [min_1, max_1]\rangle$ *et* $\langle sid_2, [min_2, max_2]\rangle$
Sortie : $\langle v, add\rangle$ *où* $v = \langle sid, [min, max]\rangle$ *et add, une valeur booléenne égale à faux si v n'a pas pu être créée.*

1. **let** $add := false$
2. **let** $v := null$
3. **if** $(sid_1 = sid_2)$ **then**
4. **find** *min the minimum element x of* $[min_2, max_2]$ *such that* $x > min_1$
5. **let** $sid := sid_1$
6. **let** $max := max_2$
7. **let** $v := \langle sid, [min, max]\rangle$
8. **let** $add := true$
9. **fi**
10. **output** $\langle v, add\rangle$

FIG. 3.9 – Algorithme de jointure temporelle locale (*LocalTpJoin*).

et $min \leq end(y) \leq max$ où $end(y)$ correspond à l'identifiant (*eid*) du dernier événement de y.

Afin de prouver la correction du comptage, nous montrons la justesse et la complétude des listes d'occurrences généralisées construites par *GoSpade*. La propriété de la justesse signifie dans ce cas que toute occurrence généralisée d'une liste, représente au moins une occurrence réelle vérifiant les contraintes. La propriété de complétude, quant à elle, signifie que pour toute occurrence réelle d'un motif, il existe une occurrence généralisée dans la liste du motif qui la représente.

Dans les démonstrations, nous allégeons les notations utilisées en simplifiant la notation adoptée pour une occurrence d'un motif et pour son préfixe et son suffixe. Ainsi, une occurrence d'un motif ne précisera plus l'identifiant des événements qui la composent. Un occurrence sera alors notée $y = e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{n-1} \rightarrow e_n$ au lieu de $y = e_1(eid(e_1)) \rightarrow e_2(eid(e_2)) \rightarrow \ldots \rightarrow e_{n-1}(eid(e_{n-1})) \rightarrow e_n(eid(e_n))$. Lorsque cela sera nécessaire, l'identifiant d'un événement e_i sera noté $t(e_i)$. Le préfixe d'une occurrence y, noté *préfixe*(y), représentera $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{n-1}$ et son suffixe (*suffixe(y)*) sera égale à e_n. Nous noterons aussi $end(y)$ l'*eid* du dernier événement y (i.e., $end(y) = t(e_n)$).

Considérons tout d'abord la définition concernant la justesse des listes d'occurrences généralisées.

Algorithme 9 (*EqJoin*)
Entrée : *GoIdList*(z_1), *GoIdList*(z_2)
Fonction appelée : *LocalEqJoin (Algorithme 10)*
Sortie : *Une nouvelle liste d'occurrences généralisées, GoIdList*

```
1. for all occ1 ∈ GoIdList(z1) do
2.   for all occ2 ∈ GoIdList(z2) do
3.       let ⟨v, add⟩ :=
           LocalEqJoin(occ1,occ2)
4.       if add then
5.         Insert v in GoIdList
6.       fi
7.   od
8. od
9. output GoIdList
```

FIG. 3.10 – Algorithme de jointure équivalente (*EqJoin*).

Définition 11 ***(justesse des listes)*** Soit S une séquence provenant d'une base de séquences β et z un motif avec sa liste d'occurrences généralisées *GoIdList*(z). *GoIdList*(z) est *juste* si pour tout v de *GoIdList*(z), où v est de la forme $\langle sid, [min, max]\rangle$ avec $sid = \mathrm{Id}(S)$, nous avons : pour tout entier t_f dans [*min*,*max*], il existe une occurrence de z dans S telle que $end(y) = t_f$.

Théorème 1 *Pour tout motif z, la liste d'occurrences généralisées, GoIdList(z), générée par GoSpade est juste.*

Preuve

Montrons que pour tout k, la liste d'occurrences généralisées, *GoIdList*, de n'importe quel k-motif z, est juste. Cette preuve est faite par induction sur k.
Soit k= 1. Par construction, les listes d'occurrences généralisées, *GoIdLists*, des items, sont justes.
Considérons maintenant $k > 1$. Supposons que les *GoIdLists* de tous les (k-1)-motifs sont *justes*.

Soit z un k-motif. Soit $v = \langle sid, [min, max]\rangle$ une occurrence généralisée provenant de *GoIdList*(z).
Nous allons montrer que $\exists S \in \beta$ telle que $sid = id(S)$ et telle que pour tout $t_{end} \in [min, max]$, il existe une occurrence y de z de la forme $e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e_n$ appartenant à S et telle que $t(e_n) = t_{end}$.

Algorithme 10 (*LocalEqJoin*)
Entrée : *Deux occurrences généralisées*
$\langle sid_1, [min_1, max_1]\rangle$ *et* $\langle sid_2, [min_2, max_2]\rangle$
Sortie : *un tuple* $\langle v, add\rangle$ *avec* $v = \langle sid, [min, max]\rangle$ *et* add, une valeur booléenne égale à faux si v ne peut pas être créée.

```
1. let add := false
2. let v := null
3. if (sid1 = sid2) then
4.    if (min1 ≤ max2 and max1 ≥ min2 ) then
5.       let sid := sid1
6.       let min := max(min1,min2)
7.       let max := min(max1,max2)
8.       let v := ⟨sid, [min, max]⟩
9.       let add := true
10.   fi
11. fi
12. output ⟨v, add⟩
```

FIG. 3.11 – Algorithme de jointure équivalente locale (*LocalEqJoin*).

GoIdList(z) a été construite par appel de la procédure *TpJoin* (algorithme 7) ou *EqJoin* (algorithme 9).

Considérons le cas où *GoIdList* a été construite par un appel à *TpJoin*.
Soit $v_1 = \langle sid_1, [min_1, max_1]\rangle$ et $v_2 = \langle sid_2, [min_2, max_2]\rangle$, deux occurrences généralisées à l'origine de la création de v. Par construction, $sid = sid_1 = sid_2$ et grâce à la justesse du niveau $k-1$, on sait qu'il existe $S \in \beta$ telle que $sid = id(S)$.

Considérons maintenant un $t_{end} \in [min,max]$. Par construction de min et max, $min \in [min_2, max_2]$ et $max \in [min_2, max_2]$. Donc, $t_{end} \in [min_2, max_2]$.

Avec l'hypothèse de justesse du niveau k-1 et puisque $t_{end} \in [min_2, max_2]$, nous pouvons affirmer qu'il existe $y_2 = e''_1 \rightarrow \ldots \rightarrow e''_{n-1}$, une occurrence de v_2 dans S telle que $t(e''_{n-1}) = t_{end}$. L'hypothèse de justesse au niveau k-1 permet aussi d'affirmer qu'il existe $y_1 = e'_1 \rightarrow \ldots \rightarrow e'_{n-1}$, une occurrence de v_1 telle que $t(e'_{n-1}) = min_1$. Par construction (algorithme *LocalTpJoin* ligne 4), $min > min_1$, et puisque $t(e''_{n-1}) = t_{end} \geq min$, nous avons $t(e'_{n-1}) < t(e''_{n-1})$.

Soit $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-1} \rightarrow e''_{n-1}$, alors y est une occurrence de z dans S, telle que end(y) = t_{end}.

Considérons maintenant le second cas, où *GoIdList* a été construite par appel de la procédure *EqJoin*(algorithme 9).
Comme pour le premier cas, considérons $v_1 = \langle sid_1, [min_1, max_1] \rangle$ et $v_2 = \langle sid_2, [min_2, max_2] \rangle$, les deux occurrences généralisées à l'origine de la construction de v. De façon similaire au premier cas, nous savons qu'il existe $S \in \beta$ telle que $sid = Id(S)$. Considérons maintenant un $t_{end} \in [min,max]$. Par construction de min et max, on a $t_{end} \in [min_1, max_1]$ et $t_{end} \in [min_2, max_2]$. L'hypothèse de justesse du niveau k-1, nous permet d'affirmer qu'il existe y_1 de la forme $e'_1 \rightarrow \ldots \rightarrow e'_{n-2} \rightarrow e'_{n-1}$, une occurrence de v_1 dans la séquence S telle que $t(e'_{n-1}) = t_{end}$. De même, il existe y_2 de la forme $e''_1 \rightarrow \ldots \rightarrow e''_{n-2} \rightarrow e''_{n-1}$, une occurrence de v_2 dans la séquence S telle que $t(e''_{n-1}) = t_{end}$. Soit $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-2} \rightarrow e_n$ où e_n est un événement composé de l'union des items de e'_{n-1} et e''_{n-1}, alors y est une occurrence de z dans S, telle que end$(y) = t_{end}$.

◇

Montrons maintenant que les listes d'occurrences généralisées sont complètes, c'est-à-dire que pour toute occurrence réelle d'un motif dans une séquence, il existe une occurrence généralisée qui la représente.

Définition 12 ***(complétude)*** Soit z un motif et *GoIdList*(z) sa liste d'occurrences généralisées. *GoIdList*(z) est *complète* si pour tout S de β et pour tout y tel que y est une occurrence de z dans S, alors il existe v dans *GoIdList*(z) telle que v *représente* y

Théorème 2 *Pour tout motif z, la liste d'occurrences généralisées, GoIdList(z), générée par GoSpade est complète.*

Preuve :

La preuve pour les k-motifs est effectuée par induction sur k.
Soit $k = 1$, par construction, les listes d'occurrences généralisées des items sont complètes.
Considérons $k > 1$ et supposons que les *GoIdLists* sont complètes pour tout les (k-1)-motifs.
Soit z un k-motif, S une séquence de β et $y = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e_n$ une occurrence de z. Le motif z possède une des formes suivantes selon le cas de génération appliqué (cf. Section 2.3.3) où p est un $(k-2)$-motif et s_1, s_2 sont de simples items : ps_1s_2 (cas de génération 1), $ps_1 \rightarrow s_2$ (cas de génération 2), $p \rightarrow s_1s_2$ (cas de génération 3), et $p \rightarrow s_1 \rightarrow s_2$ (cas de génération 4). Les formes $p \rightarrow s_2 \rightarrow s_1$ (cas de génération 5) et $p \rightarrow s_1 \rightarrow s_1$ (cas de génération 6) sont similaires à la forme $p \rightarrow s_1 \rightarrow s_2$ pour cette démonstration.

Tout d'abord, considérons le cas où z est de la forme $p \rightarrow s_1 \rightarrow s_2$ (cas de génération 4). Alors, une occurrence y de z est de la forme $e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e_n$ où e_{n-1} et e_n sont des singletons ($e_{n-1} = \{s_1\}$ et $e_n = \{s_2\}$).

Soit $y_1 = e_1 \rightarrow \ldots \rightarrow e_{n-1}$ et $y_2 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_n$. Alors, y_1 (resp. y_2) est une occurrence d'un (k-1)-motif z_1 (resp. z_2) dans S. L'hypothèse de complétude du niveau k-1, nous permet d'affirmer qu'il existe v_1 provenant de $GoIdList(z_1)$ (resp. il existe v_2 provenant de $GoIdList(z_2)$) telle que y_1 (resp. y_2) soit représentée par v_1 (resp. v_2). Soit v_1 et v_2 notées respectivement $\langle sid_1, [min_1, max_1]\rangle$ et $\langle sid_2, [min_2, max_2]\rangle$.

Nous montrons maintenant que durant la construction de $GoIdList(z)$, obtenue en effectuant un appel à $TpJoin$ (cas de génération 4) entre $GoIdList(z_1)$ et $GoIdList(z_2)$, une occurrence généralisée v a été construite à partir de v_1 et de v_2 et qu'elle est telle que v représente y.

Nous avons $t(e_{n-1}) \geq min_1$ (puisque v_1 représente y_1), $t(e_n) \leq max_2$ (puisque v_2 représente y_2) et $t(e_{n-1}) < t(e_n)$ (e_{n-1} et e_n sont des événements successifs dans y). Donc $min_1 < max_2$. De plus, $sid_1 = sid_2$ (y_1 et y_2 sont des occurrences dans S). Donc, une occurrence v a été générée à partir de v_1 et de v_2 (tests ligne 3 algorithme 7 et ligne 3 algorithme 8). Par construction $v = \langle sid, [min, max]\rangle$ où $sid = sid_1$, $max = max_2$ et min est l'élément minimum $x \in [min_2, max_2]$ tel que $x > min_1$.
Montrons maintenant que cette occurrence généralisée v, représente y, i.e. $sid = Id(S)$ et $min \leq t(e_n) \leq max$. La première propriété, $sid = id(S)$ est satisfaite parce que $sid = sid_1$ et $sid_1 = id(S)$ (car v_1 représente y_1). Puisque v_2 représente y_2, $t(e_n) \leq max_2$, et puisque $max = max_2$, nous avons $t(e_n) \leq max$. Pour montrer que $t(e_n) \geq min$, nous savons que $t(e_{n-1}) \geq min_1$ et que $t(e_n) > t(e_{n-1})$, ainsi $t(e_n) > min_1$. De plus, nous avons $t(e_n) \in [min_2, max_2]$ (puisque v_2 représente y_2). Par construction, nous savons que min est l'élément minimum de $[min_2, max_2]$ supérieur à min_1. D'où $t(e_n) \geq min$.

Ainsi, nous avons prouvé qu'une occurrence généralisée v a été construite à partir de v_1 et de v_2 et que cette occurrence généralisée est telle que v représente y. Ceci prouve la complétude des $GoIdLists$ construites par un appel à $TpJoin$ dans le cas de génération 4.

La preuve pour le cas de génération 2, c'est-à-dire le cas où z est de la forme $ps_1 \rightarrow s_2$ est similaire au cas précédent, où e_n est un singleton ($e_n = \{s_2\}$) mais où e_{n-1} est un événement composé de s_1 et d'au moins un autre item.
Dans ce cas, soit $y_1 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1}$ et $y_2 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e'_{n-1} \rightarrow e_n$ telles que $t(e'_{n-1}) = t(e_{n-1})$ et $e'_{n-1} = e_{n-1} \setminus \{s_1\}$. y_1 (resp. y_2) est une occurrence du motif z_1 (resp. z_2) dans S. Alors, le même raisonnement que pour le cas de génération précédent peut être appliqué pour montrer qu'une occurrence généralisée v est construite lors d'un appel à $TpJoin$ entre $GoIdList(z_1)$ et $GoIdList(z_2)$ et que v représente y.

Considérons maintenant le cas où z est de la forme $p \rightarrow s_1 s_2$ (cas de génération 3). Alors, l'occurrence y de z dans S est de la forme $e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e_n$ où e_n est un événement composé de deux items s_1 et s_2.
Soit $y_1 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e'_n$ et $y_2 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e''_n$, où $e'_n = \{s_1\}$ et $e''_n = \{s_2\}$ et $t(e_n) = t(e'_n) = t(e''_n)$. y_1 (resp. y_2) est une occurrence du motif z_1 (resp. z_2) dans S.
L'hypothèse de complétude du niveau k-1, nous permet d'affirmer qu'il existe v_1 dans $GoIdList(z_1)$ et qu'il existe v_2 dans $GoIdList(z_2)$ telles que y_1 et y_2 sont respectivement représentées par v_1 et v_2. Soit v_1 et v_2 de la forme $\langle sid_1, [min_1, max_1]\rangle$ et $\langle sid_2, [min_2, max_2]\rangle$.
Nous montrons qu'une occurrence généralisée v représentant y a été construite à partir de v_1 et de v_2 lors de la construction de $GoIdList(z)$, par appel à $EqJoin$ sur $GoIdList(z_1)$ et $GoIdList(z_2)$.
Puisque v_1 et v_2 représentent respectivement y_1 et y_2 nous avons $t(e'_n) \in [min_1, max_1]$ et $t(e''_n) \in [min_2, max_2]$. Puisque $t(e_n) = t(e'_n) = t(e''_n)$, alors $t(e_n) \in [min_1, max_1] \cap [min_2, max_2]$. Ainsi $[min_1, max_1] \cap [min_2, max_2]$ n'est pas vide. De plus, $sid_1 = sid_2$ puisque y_1 et y_2 sont des occurrences dans S. Ainsi, les tests des lignes 3 et 4 de l'algorithme 10 sont vérifiés et une occurrence généralisée $v = \langle sid, [min, max]\rangle$ est ajoutée à $GoIdList(z)$ avec $sid = sid_1$ et par construction $[min,max] = [min_1, max_1] \cap [min_2, max_2]$. Enfin, puisque $t(e_n) \in [min_1, max_1] \cap [min_2, max_2]$ nous avons $t(e_n) \in [min,max]$. Comme $sid = sid_1 = Id(S)$ alors nous savons que v représente y.

Enfin, considérons le dernier cas possible où z est de la forme $ps_1 s_2$ (cas de génération 1). L'occurrence y de z dans S est alors de la forme $e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e_n$ où e_n est un événement composé d'au moins trois items et contenant s_1 et s_2.
Soit $y_1 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e'_n$ et $y_2 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e''_n$, où $e'_n = e_n \setminus \{s_2\}$, $e''_n = e_n \setminus \{s_1\}$ et $t(e_n) = t(e'_n) = t(e''_n)$. Le même raisonnement que dans le cas précédent peut alors être appliqué pour montrer qu'une occurrence généralisée de z représentant y est aussi construite.

◇

Le théorème suivant découle directement des théorèmes 1 et 2.

Théorème 3 ***(correction)*** *Pour tout motif z, le support déterminé par GoSpade à partir de d'une liste d'occurrences généralisées (GoIdList), est le même que le support obtenu par Spade à partir d'une liste d'occurrences (IdList).*

3.2.4 Résultats expérimentaux

Dans cette section, nous présentons des résultats expérimentaux montrant que le comportement de l'algorithme *Spade* est largement amélioré par l'utilisation

d'occurrences généralisées lorsque les bases de données contiennent des répétitions consécutives. Les algorithmes *GoSpade* et *Spade* ont été développés sous Microsoft Visual C++ 6.0, en effectuant le même niveau d'optimisation pour permettre leur comparaison. Toutes les exécutions ont été effectuées sur un PC ayant 196 MB de mémoire et un processeur Pentium III tournant à 500 MHz sous Microsoft Windows 2000.

Les exécutions ont été effectuées sur un jeu de donnée synthétique généré en utilisant le générateur Dataquest d'IBM[1] [AS94b]. Le jeu $set1$ a ainsi été généré en utilisant avec les paramètres suivants (décrits dans [SA96]) : C10-T2.5-S4-I1.25-D1K sur un alphabet constitué de 100 items. Ce jeu contient 1000 séquences (D = 1K, i.e. 1000), dont la longueur moyenne a été fixée à 10 événements (C = 10). Dans ce jeu de données, l'unité de temps est fixée à 1 et chaque unité de temps possède exactement un événement. Ces premières expérimentations sont effectuées sur un jeu relativement réduit, leur objectif étant d'analyser l'impact des occurrences généralisées sur la taille des listes, les coûts de jointure et les temps d'exécution. Nous donnerons dans la section 4.1 des résultats portant sur des jeux de tailles plus importantes (de 20 000 à 100 000 séquences), obtenus en utilisant notre prototype final (*GoSpade* étendu à la gestion des contraintes et incluant des optimisations spécifiques).

Afin d'avoir des jeux de données présentant des répétitions consécutives de certains items, nous avons effectué un post-traitement paramétré sur le jeu $set1$. Chaque item rencontré dans un événement d'une séquence a une probabilité fixée à 10% d'être répété. Nous fixons un paramètre entier i et, lorsqu'un item est répété, nous le répétons simplement dans les i événements consécutifs. Si la fin d'une séquence est atteinte durant le processus de duplication, nous n'ajoutons pas de nouvel événement à la séquence et l'item courant n'est alors pas complètement dupliqué. Nous notons $set1\_r\{i\}$ le jeu obtenu avec un paramètre de répétition fixé à i. Dans un soucis d'uniformité, le jeu initial $set1$ est noté $set1\_r0$. Le post-traitement de $set1\_r0$ amène à la création de 5 nouveaux jeux $set1\_r1$,..., $set1\_r5$. Tous ces jeux ont été créés simultanément afin que les mêmes items soient répétés dans les différents jeux. Par exemple, si l'item A est un item à répéter et qu'il apparaît dans une séquence à l'instant 5, alors il sera ajouté à l'événement se trouvant à l'instant 6 dans le jeu $set1\_r1$, aux événements se trouvant aux instants 6 et 7 dans le jeu $set1\_r2$, et ainsi de suite.

Impact des occurrences généralisées sur la taille des listes

Les occurrences généralisées représentent d'une façon compacte toutes les occurrences consécutives qui peuvent être présentes dans une base de séquences. La figure 3.12 montre les tailles des listes d'occurrences de *Spade* (*IdLists*) et de *GoSpade* (*GoIdLists*) (en terme de nombre d'éléments) pour des extractions effectuées sur les jeux $set1\_r0$, $set1\_r1$, ..., $set1\_r5$ en utilisant différents seuils

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

de supports de fréquence minimum (7.5%, 5% et 2.5%). Le nombre d'occurrences utilisées par *Spade* est supérieur au nombre d'occurrences généralisées utilisées par *GoSpade*. Comme nous pouvions l'imaginer, cette réduction augmente avec le nombre de répétitions consécutives.

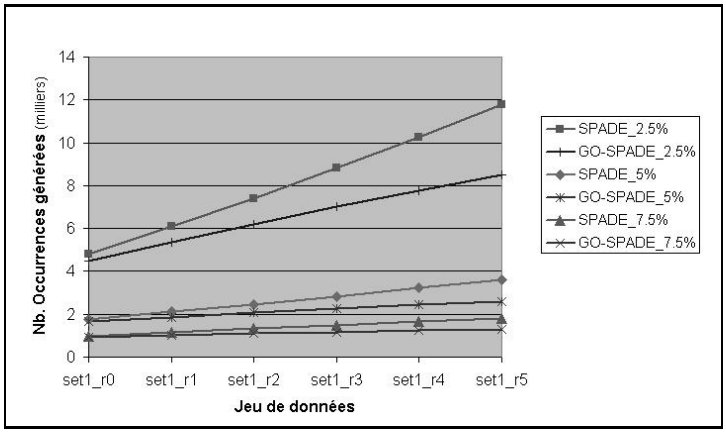


FIG. 3.12 – Evolution du nombre total d'occurrences utilisées.

Impact des listes d'occurrences généralisées sur les coûts de jointure

Comme il vient d'être montré, les tailles des listes d'occurrences généralisées sont plus petites que les tailles des listes d'occurrences de *Spade*. Cette réduction a un impact direct sur le coût des jointures. En effet, soit n_s et m_s, le nombre d'occurrences de deux motifs générateurs dans une séquence s. Dans le pire des cas, et en supposant qu'il y a $nbSeq$ séquences dans la base, le nombre de comparaisons nécessaires pour effectuer une jointure entre leurs deux listes est de $\sum_s n_s m_s$, $s \in [1, \ldots, nbSeq]$. Supposons maintenant que n_s et m_s soient réduits d'un facteur moyen de $\gamma \leq 1$. Dans ce cas, le nombre de comparaisons devient $\sum_s \gamma^2 n_s m_s$, $s \in [1, \ldots, nbSeq]$, ce qui signifie que le nombre de comparaisons pour *GoSpade* est réduit par un facteur de γ^2 par rapport à *Spade*.

La figure 3.13 montre cette réduction en pratique sur des extractions effectuées sur les jeux $set1\_r0$ à $set1\_r5$, avec les seuils de fréquences absolues fixées à 2.5% et à 7.5%. Par exemple, le coût en terme de nombre de comparaisons nécessaires à une extraction pour *GoSpade* à un seuil de 2.5% est significativement moins important que le coût de *Spade* à ce même support. On peut même remarquer que ce coût est proche de celui de de *Spade* à un seuil de 7.5%, c'est-à-dire à un seuil plus fort.

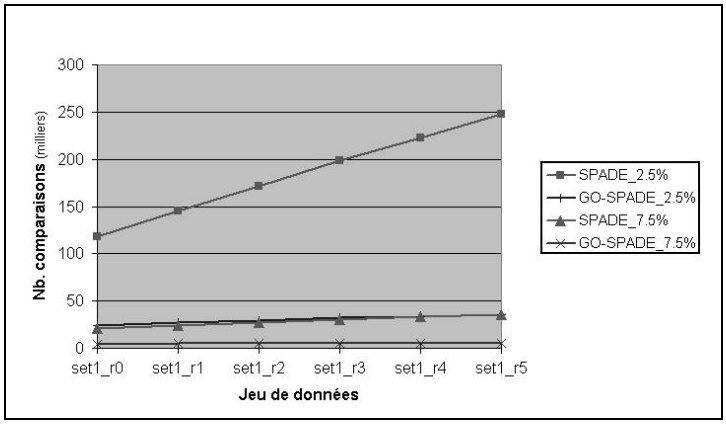


FIG. 3.13 – Evolution du nombre total de comparaisons.

Impact des listes d'occurrences généralisées sur le temps d'exécution

La réduction de la taille des listes et la réduction du nombre de comparaisons permet de réduire considérablement le temps d'exécution total des extractions. Ce fait est illustré sur la figure 3.14, qui représente les temps d'exécution obtenus par *Spade* et par *GoSpade* pour des extractions effectuées sur les jeux $set1\_r0$ à $set1\_r5$ aux seuils respectifs de 2.5%, 5% et 7.5%.

Dans la figure 3.3 (Section 3.1), nous avions montré que le temps nécessaire à *Spade*, pour extraire un certain nombre de motifs, augmentait en présence de séquences contenant des répétitions consécutives. La figure 3.15 complète ces résultats en précisant le temps nécessaire à *GoSpade*. Cette figure montre que le temps d'exécution nécessaire à *GoSpade* ne varie que très peu en présence de répétitions consécutives.

Les résultats expérimentaux que nous venons de présenter ont permis de montrer que les gains obtenus sur la taille des listes réduisaient grandement les temps d'exécution lorsque les bases contiennent des répétitions consécutives. Dans la pratique il est intéressant d'introduire des contraintes autres que celle de la fréquence minimum afin d'être plus sélectif quant aux motifs produits tout en réduisant l'espace de recherche.

La section suivante présente les modifications à apporter aux occurrences généralisées afin que celles-ci puissent être utilisées lors d'extractions faisant intervenir des contraintes temporelles (e.g., contrainte de fenêtre de temps maximum). Un nouvel algorithme, adapté à la gestion de ces occurrences généralisées ainsi complétées et prenant en compte ce type de contraintes est alors proposé.

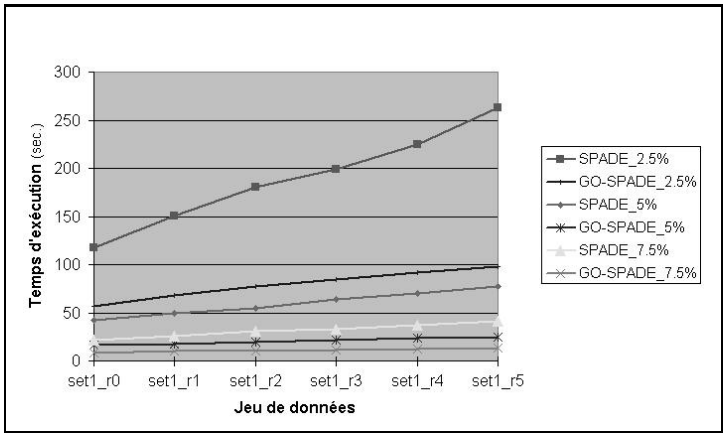


FIG. 3.14 – Evolution du temps d'exécution total.

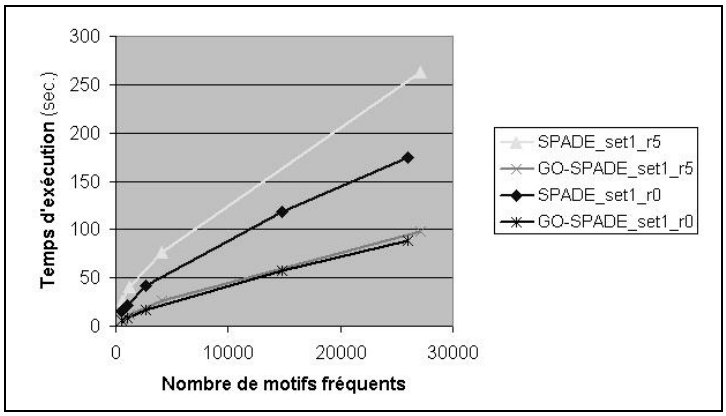


FIG. 3.15 – Influence des répétitions consécutives sur *Spade* et *GoSpade*.

3.3 L'algorithme *GoSpec*

L'utilisation d'une représentation par listes d'occurrences généralisées peut être complétée afin d'introduire la gestion de contraintes de temps (contrainte de fenêtre de temps maximum, contrainte de gap minimum et contrainte de gap maximum, cf. section 2.2.2).

Dans cette section, nous présentons l'algorithme *GoSpec* [LRBE03a] (Generalized Occurrences for Sequential Pattern Extraction under Constraints), basé sur l'algorithme *GoSpade* [LRBE03b] et qui permet d'introduire des contraintes de temps. Ce dernier peut aussi gérer certaines contraintes syntaxiques, mais ces dernières ne seront pas intégrées ici, puisqu'elles peuvent être gérées de façon similaire à *cSpade*.

Les modifications apportées à cette version contrainte portent d'une part sur la structure des occurrences généralisées utilisées (section 3.2.1), et d'autre part sur les opérations de jointure appliquées. Les occurrences généralisées doivent en effet être complétées afin de pouvoir prendre en compte les contraintes de temps concernées : contrainte de fenêtre maximum, contrainte de gap minimum et contrainte de gap maximum.

Dans la section suivante nous présentons la structure des nouvelles occurrences généralisées, appelées *occurrences généralisées contraintes*, ainsi que les algorithmes de jointure associés à la gestion de listes d'occurrences généralisées contraintes. Nous fournissons ensuite une preuve de la correction de ces algorithmes de jointure et nous terminons avec une présentation de résultats expérimentaux montrant les gains de performance de *GoSpec* par rapport à l'algorithme *cSpade* [Zak00].

3.3.1 Les occurrences généralisées contraintes

La structure d'une liste d'occurrences généralisées contraintes permet, comme les listes d'occurrences généralisées utilisées par *GoSpade*, de réduire la taille des listes en représentant plusieurs occurrences par une seule occurrence plus générale.

Les occurrences généralisées contraintes viennent compléter les informations contenues dans les occurrences généralisées (identifiant de séquence, dates d'apparitions consécutives du dernier événement) afin de pouvoir prendre en compte les contraintes de temps, comme la contrainte de fenêtre maximum ou encore la contrainte de gap maximum.

Une occurrence généralisée contrainte se compose des informations suivantes :

pour un motif z, une occurrence généralisée contrainte correspond à un tuple $\langle sid, tBeg, [min, max], gmax \rangle$, composé des informations suivantes :

- Un identifiant sid correspondant à l'identifiant d'une séquence dans laquelle le motif z apparaît.

- Une valeur $tBeg$ correspondant à la date d'apparition du premier événement du motif z.
- Un intervalle $[min,max]$ correspondant aux $eids$ des occurrences consécutives du dernier événement du motif z.
- Une valeur $gmax$ indiquant la date d'apparition de la dernière occurrence du dernier événement du motif z respectant la contrainte de gap maximum. Si cette occurrence n'existe pas, alors $gmax$ est fixé à -1.

Ainsi, une occurrence généralisée contrainte diffère d'une occurrence généralisée par l'ajout de deux champs supplémentaires, $tBeg$ et $gmax$. Chacun de ces champs contribue à la prise en compte d'une contrainte de temps spécifique : la contrainte de fenêtre maximum pour $tBeg$ et la contrainte de gap maximum pour $gmax$. Au niveau des motifs de taille 1, les informations apportées par $tBeg$ et $gmax$ ne sont pas significatives puisque les contraintes de temps ne s'appliquent pas sur les items. Cependant, ce n'est pas le cas pour les motifs plus longs, et nous examinerons par la suite le rôle respectif de $tBeg$ et $gmax$ au niveau des motifs de longueur > 1.

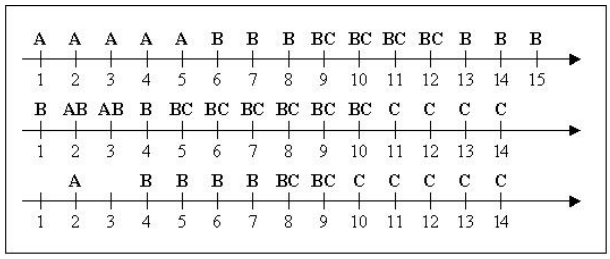


FIG. 3.16 – Base de séquences contenant des répétitions consécutives.

Pour le moment, étudions le contenu d'une occurrence généralisée contrainte pour des motifs de la base exemple (figure 3.16). Leur liste d'occurrences sont représentées dans la figure 3.17. Pour chacun des motifs, la table de gauche représente sa liste d'occurrences au sens de *cSpade*, et celle de droite correspond à sa liste d'occurrences généralisées contraintes. Ainsi, dans le cas de l'item B, on note que la représentation par une liste d'occurrences généralisées contraintes permet de réduire ses 10 occurrences consécutives dans la première séquence à seulement une occurrence généralisée contrainte $\langle 1,\ 6,\ [6,\ 15],\ 15\rangle$, où l'intervalle [6,15] compacte ces 10 $eids$.

Considérons maintenant la dernière occurrence généralisée contrainte de la liste du motif A $\rightarrow$ B. Cette occurrence généralisée, $\langle 3,\ 2,\ [4,\ 9],\ 7\rangle$, indique que ce motif apparaît dans la troisième séquence de la base et qu'il commence à la date 2. L'intervalle [4,9] signifie que cette occurrence représente toutes les occurrences de ce motif se terminant à tous les instants entre 4 et 9. La valeur de $gmax$ égale

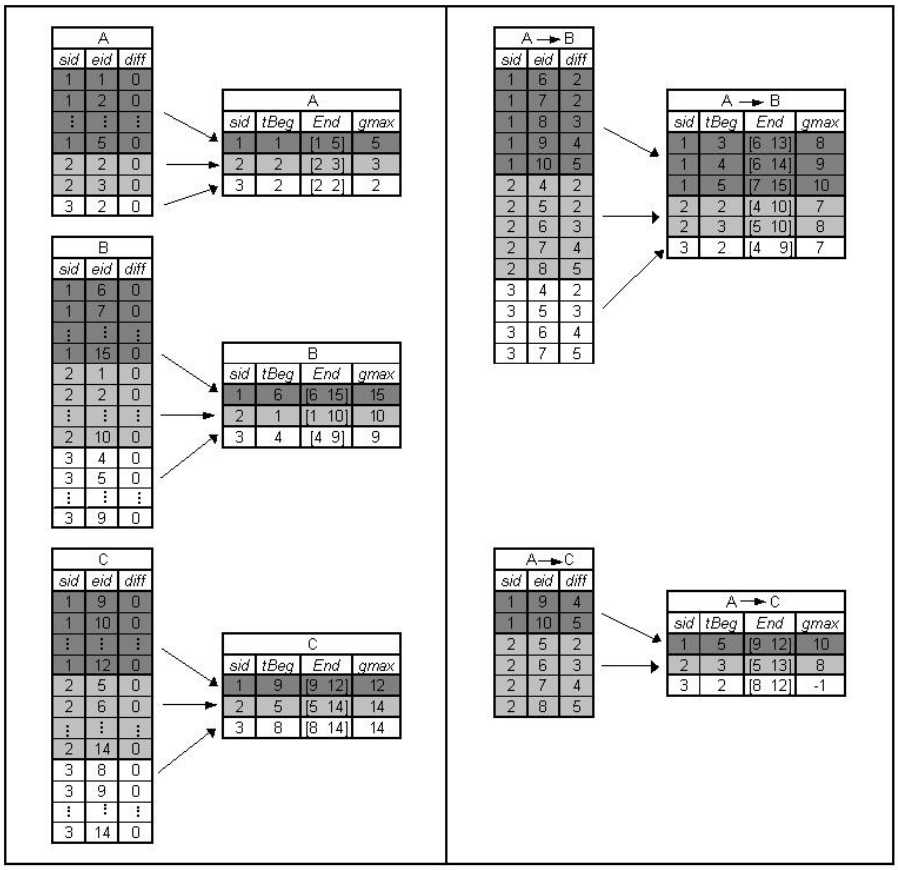


FIG. 3.17 – Listes d'occurrences et listes d'occurrences généralisées contraintes pour les motifs A, B, C, A $\rightarrow$ B, A $\rightarrow$ C et A $\rightarrow$ B $\rightarrow$ C de la base de la figure 3.16, avec les contraintes suivantes : $minSup = 2$, $minGap = 2$, $maxGap = 5$ et $maxWin = 10$.

à 7 signifie que les occurrences du motif se terminant entre les instants 4 et 7 satisfont entièrement la contrainte de gap maximum (fixée à 5). Les occurrences de ce motif se terminant après la date 7, ne satisfont cette contrainte que sur leur préfixe, c'est-à-dire entre tous leurs événements du préfixe, mais pas entre le dernier événement du préfixe et le suffixe.

L'information apportée par $tBeg$:

Une contrainte de fenêtre maximum fixée à $winMax$ est vérifiée pour une occurrence d'un motif de la forme $\alpha_1(eid(\alpha_1)) \rightarrow \alpha_2(eid(\alpha_2)) \rightarrow \ldots \rightarrow \alpha_n(eid(\alpha_n))$, si $eid(\alpha_n) - eid(\alpha_1) \leq winMax$. Dans une occurrence généralisée contrainte, la valeur de $tBeg$ correspond à $eid(\alpha_1)$ et l'intervalle $[min,max]$ contient toutes les valeurs possibles de $eid(\alpha_n)$ telles que $min \leq eid(\alpha_n) \leq max$. Dans ce cas, les occurrences représentées par une occurrence généralisée contrainte couvrent des tailles de fenêtres appartenant à l'intervalle $[min - tBeg,max - tBeg]$, la fenêtre de temps maximum étant obtenue par l'occurrence du motif dont $eid(\alpha_n) = max$. Par exemple, les occurrences du motif A $\rightarrow$ B dans la troisième séquence de la base, représentées par l'occurrence généralisée contrainte $\langle 3, 2, [4, 9], 7\rangle$, couvrent toutes les tailles de fenêtres entre 2 (i.e., 4-2) et 7 (i.e., 9-2).

Ainsi, une contrainte de fenêtre maximum sera vérifiée pour toutes les occurrences représentées par une occurrence généralisée contrainte, si l'occurrence possédant la fenêtre maximum respecte cette contrainte, c'est-à-dire si $max - tBeg \leq winMax$. Par exemple, toutes les occurrences du motif A $\rightarrow$ B dans la troisième séquence vérifient la contrainte de fenêtre fixée à 10 puisque la fenêtre maximum est de $7 < 10$.

Si nous avions voulu représenter toutes les occurrences d'un motif dans une séquence, il nous aurait fallu considérer toutes les dates de début possible de ce motif, tout comme nous avons considéré toutes ses dates de fin. Par exemple, dans la première séquence, le motif A $\rightarrow$ B possède des occurrences débutant aux instants consécutifs 1,2,3,4 et 5. Cependant, dans le cas de la seule présence d'une contrainte de fenêtre maximum, nous pouvons conserver qu'une seule date de début possible et celle-ci correspond à la date de début au plus tard du motif (i.e, 5). Justifions maintenant ce choix de $tBeg$.

Considérons pour cela toutes les occurrences du motif A $\rightarrow$ B, en se limitant à la seule contrainte de fenêtre maximum fixée à 10 dans notre exemple. Dans ce contexte, considérons toutes les occurrences de ce motif dans la première séquence et débutant à l'instant 1. Ces occurrences sont au nombre de 6 : A (1) $\rightarrow$ B(6), A(1) $\rightarrow$ B(7), ...,A(1) $\rightarrow$ B(11). Les occurrences se terminant au delà (A(1) $\rightarrow$ B(12), ..., A(1) $\rightarrow$ B(15)) ne sont plus considérées puisqu'elles ne respectent pas la contrainte de fenêtre (15-1 > 14-1 > 13-1 > 12-1 > $winMax$). Considérons maintenant toutes les occurrences de ce motif commençant à l'instant 2 (séquence 1). Ces occurrences sont au nombre de 7 : A(2) $\rightarrow$ B(6), A(2) $\rightarrow$ B(7), ..., A(2)

$\rightarrow$ B(12). On remarque que ces occurrences couvrent les mêmes dates de fin que celles couvertes par une date de début fixée à 1, c'est-à-dire les occurrences se terminant à tous les *eids* entre 6 et 11, mais qu'en plus elle permet de couvrir une date de fin supplémentaire ($eid = 12$) puisque celle-ci respecte alors la contrainte de fenêtre de temps (12-2 = 10). Si on considère les autres dates de début du motif, c'est-à-dire celles correspondant aux *eids* 3, 4 et 5, on remarque que le même phénomène se reproduit, c'est-à-dire que plus on décale la date de début considérée, plus l'intervalle de temps couvert par les dates de fin possibles du motif augmente. Ainsi, le nombre d'occurrences maximum respectant la contrainte de fenêtre sera obtenu lorsque la date de début considérée sera égale à 5 (i.e., la date de début au plus tard).

Ainsi, le choix de ne considérer qu'une seule date pour la date de début d'un motif et que cette date corresponde à la date de début au plus tard se justifie par le fait que la date de début au plus tard est suffisante pour considérer le maximum d'occurrences d'un motif dans une séquence.

Cependant, dans certains cas, il est nécessaire de conserver plusieurs dates de début possibles, ce qui se traduit par une multiplication des occurrences généralisées contraintes, chacune débutant à une date de début différente. Cela se produit lorsque l'on considère une contrainte de *gap minimum* en plus de la contrainte de fenêtre. En effet, considérons de nouveaux les occurrences du motif A $\rightarrow$ B, en faisant intervenir, en plus de la contrainte de fenêtre fixée à 10, une contrainte de gap minimum, $gapMin$, fixée à 2. Considérons toutes les occurrences de ce motif commençant à la date 5. Ces occurrences sont au nombre de 10 : A(5) $\rightarrow$ B(6), A(5) $\rightarrow$ B(7), ..., A(5) $\rightarrow$ B(15) et respectent toutes la contrainte de fenêtre fixée. Toutes ces occurrences respectent aussi la contrainte de gap minimum, sauf la première occurrence, A(5) $\rightarrow$ B(6), dont le gap n'est que de 1 (6-5), alors qu'il existe bien une occurrence se terminant à cette date et qui vérifie la combinaison de ces contraintes, par exemple l'occurrence A(4) $\rightarrow$ B(6). Dans un souci de complétude, il faut donc considérer la date de début 4, qui permet de couvrir toutes les dates de fin possibles hormis la date de fin 15 (A(4) $\rightarrow$ B(15) ne respecte pas la contrainte de fenêtre). Mais il faut aussi conserver l'occurrence généralisée correspondant à la date de début 5 puisque celle-ci permet de couvrir un maximum de dates de fin pour le motif excepté l'occurrence se terminant à la date 6. Ce phénomène doit être traité de façon individuelle lors de la génération des motifs de taille 2, puisque les dates de début des items seuls (motifs de taille 1) n'intègrent pas cet aspect. Cela se répercute automatiquement lors de la génération des motifs de taille supérieure à 2, puisque les motifs de taille 2 intégreront les différentes dates de début.

L'information apportée par *gmax* :

Le champs *gmax* contient l'information nécessaire à la contrainte de gap maxi-

mum. En effet, il permet de connaître toutes les occurrences d'un motif représentées par une occurrence généralisée contrainte qui vérifient la contrainte de gap maximum. Ainsi, toutes les occurrences dont la date de fin est inférieure à la valeur de *gmax*, vérifient cette contrainte, et celles dont la date de fin est supérieure ne la vérifient pas. Par exemple, les occurrences du motif A $\rightarrow$ B se terminant aux instants 8 et 9 dans la troisième séquence de la base ne vérifient pas la contrainte de gap maximum fixée à 5 (9-2 > 8-2 > 5).
Lorsqu'aucune occurrence d'un motif ne vérifie la contrainte, la valeur du *gmax* est fixée à -1. C'est par exemple le cas pour l'occurrence généralisée contrainte $\langle 3, 2, [8, 12], -1 \rangle$ qui représente les occurrences du motif A $\rightarrow$ C dans la troisième séquence de la base. En effet, aucune de ces occurrences ne vérifie la contrainte (12-2 > 11-2 > ... > 8-2 > 5).

Cependant, l'information concernant les occurrences ne vérifiant pas la contrainte de gap maximum doit être conservée, puisqu'elle est nécessaire à la génération d'autres motifs, comme celle du motif A $\rightarrow$ B $\rightarrow$ C. En effet, le fait qu'une occurrence de B apparaisse entre A et C permet de ne pas avoir à s'occuper de la vérification de la contrainte entre A et C mais uniquement entre A et B (cette information est fournie par la liste d'occurrences du préfixe, ici le motif A $\rightarrow$ B) et entre B et C (c'est-à-dire entre le dernier événement du préfixe et le suffixe).

Dans la partie suivante, nous décrivons les opérations de jointures adaptées à la gestion des occurrences généralisées contraintes, l'algorithme général de *GoSpec* restant le même que *GoSpade* . Leur justesse et leur complétude sont montrées en partie 3.3.3. Les gains obtenus par l'utilisation d'une représentation par listes d'occurrences généralisées contraintes sont présentés et analysés dans la partie 3.3.4.

3.3.2 Les opérations de jointure appliquées par *GoSpec*

Une opération de jointure est appelée après avoir effectué une opération de fusion entre deux motifs générateurs, z_1 et z_2. Elle permet de construire la liste d'occurrences généralisées contraintes d'un motif candidat z à partir des *occLists* de ses deux motifs générateurs z_1 et z_2. Comme pour l'algorithme *GoSpade*, il existe deux types de jointures, les jointures temporelles et les jointures équivalentes. L'opération de jointure à appliquer dépend du cas de génération dont le motif candidat découle. Les différents cas de génération utilisés par l'algorithme *GoSpec* correspondent aux cas appliqués par *Spade* (section 2.3.3) et réutilisés par l'algorithme *GoSpade* (section 3.2).

La jointure temporelle contrainte

Pour effectuer cette opération, deux procédures différentes peuvent être appelées selon le niveau auquel se trouve le processus d'extraction : $cTpJoinLevel_2$

(Algorithme 13) et *cTpJoin* (Algorithme 11). La première procédure est un algorithme spécifique dédié au cas particulier des motifs candidats de taille 2 et la deuxième correspond au cas général des k-motifs candidats lorsque $k > 2$. Ces deux algorithmes font appel à une fonction commune appelée *cLocalTpJoin* (Algorithme 12) qui construit une occurrence généralisée contrainte $v = \langle sid, tBeg, [min, max], gmax \rangle$ d'un motif z à partir de deux occurrences généralisées individuelles provenant respectivement des listes de z_1 et de z_2.

Algorithme 11 (*cTpJoin*)
Entrée : *occList(z_1) et occList(z_2), les listes d'occurrences généralisées contraintes de deux motifs générateurs.*
Fonction appelée : *Algorithme 12 (cLocalTpJoin).*
Sortie : *Une nouvelle liste d'occurrences généralisées contraintes, occList.*

Initialiser GoIdList à la liste vide.
1. **for all** $occ_1 \in GoIdList(z_1)$ **do**
2. **for all** $occ_2 \in GoIdList(z_2)$ **do**
3. **let** $\langle v, add \rangle := cLocalTpJoin(occ_1, occ_2)$
4. **if** *add* **then**
5. *Insert v in occList*
6. **fi**
7. **od**
8. **od**
9. **output** *occList*

FIG. 3.18 – Algorithme de jointure temporelle contrainte (*cTpJoin*).

La fonction *cLocalTpJoin* (Algorithme 12), commence par vérifier (ligne 3) que les occurrences généralisées contraintes fournies en entrée satisfont les conditions nécessaires afin de pouvoir effectuer leur jointure. La dernière condition (i.e., $min_1 \leq gmax_1$), provient de la valeur -1 attribuée à $gmax_1$ dans le cas où aucune occurrence n'est représentée par $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1 \rangle$ et permet de s'assurer qu'il existe au moins une occurrence représentée par $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1 \rangle$ qui vérifie la contrainte de gap maximum.
Afin d'illustrer le principe des deux premières conditions, considérons l'exemple suivant :

Exemple 11 *Soient deux occurrences généralisées, $cOccGen_1$ du motif $A \rightarrow B$ et $cOccGen_2$ du motif $A \rightarrow C$, représentées sur la figure 3.20. Ces occurrences généralisées ont respectivement pour date de début $tBeg_1$ et $tBeg_2$ et pour intervalles de fin $[min_1, max_1]$ et $[min_2, max_2]$, et correspondent à des occur-*

Algorithme 12 (*cLocalTpJoin*)
Entrée : *Deux occurrences généralisées contraintes*
$\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$
et $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$
Sortie : $\langle v, add\rangle$, *où* $v = \langle sid, tBeg, [min, max], gmax\rangle$
et add *est une valeur booléenne égale à faux si v n'a pu être créée.*

1. **let** $add := false$
2. **let** $v := null$
3. **if** $(min_1 + gapMin \leq max_2)$
$and\ (tBeg_1 + winMax \geq min_2)$
$and\ (min_1 \leq gmax_1)$**then**
4. **if** $(sid_1 = sid_2)$ **then**
5. **let** $sid := sid_1$
6. **let** $tBeg := tBeg_1$
7. **trouver** *min, l'élément minimum x de* $[min_2, max_2]$
tel que $x \geq min_1 + gapMin$
8. **trouver** *max, l'élément maximum x de* $[min_2, max_2]$
tel que $x \leq tBeg_1 + winMax$
9. **trouver** *gmax, l'élément maximum x de* $[min_2, max_2]$
tel que $x \leq gmax_1 + gapMax$
10. **fi**
11. **if** *(min et max existent) et* $(min \leq max)$ **then**
12. **if** *(gmax n'existe pas)* **then let** $gmax := -1$
13. **else if** $(gmax > max)$ **then**
14. **let** $gmax := max$ **fi**
15. **fi**
16. **let** $v := \langle sid, tBeg, [min, max], gmax\rangle$
17. **let** $add := true$
18. **fi**
*19.***fi**
*20.***output** $\langle v, add\rangle$

FIG. 3.19 – Algorithme de jointure temporelle locale contrainte (*cLocalTpJoin*).

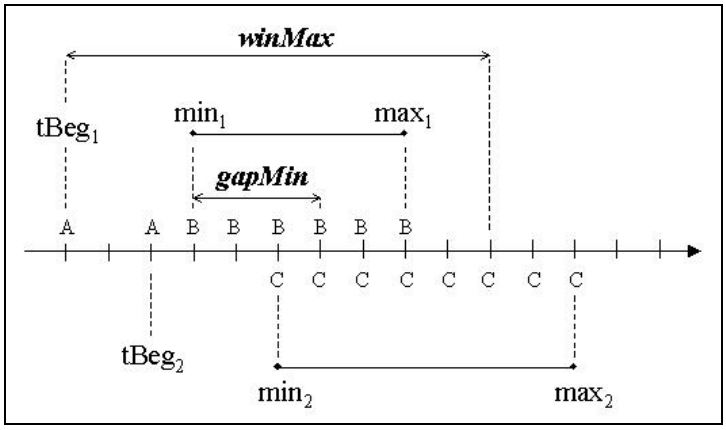


FIG. 3.20 – Illustration des tests d'entrée de l'algorithme 12 ($cLocalTpJoin$).

rences respectant une contrainte de fenêtre, $winMax$, et une contrainte de gap minimum, $gapMin$. Considérons les occurrences du motif $A \rightarrow B \rightarrow C$ pouvant être construites à partir des occurrences de $A \rightarrow B$ représentées par $cOccGen_1$ et de $A \rightarrow C$ représentées par $cOccGen_2$, et plus particulièrement les occurrences représentées par une nouvelle occurrence généralisée débutant en $tBeg_1$.

Une condition nécessaire pour qu'il existe une telle occurrence de $A \rightarrow B \rightarrow C$ vérifiant la contrainte de gap minimum, $gapMin$, est qu'il existe une occurrence de $B \rightarrow C$ vérifiant la contrainte de gap minimum, c'est-à-dire que $min_1 + gapMin \leq max_2$.

Une seconde condition nécessaire pour qu'il existe une telle occurrence de $A \rightarrow B \rightarrow C$ vérifiant la contrainte de fenêtre maximum, $winMax$, est qu'il existe une occurrence de $A \rightarrow C$ vérifiant aussi cette contrainte avec une date de début égale à $tBeg_1$, c'est-à-dire que $tBeg_1 + winMax \geq min_2$.

Lorsque ces trois conditions nécessaires ont été vérifiées, et après avoir vérifié (ligne 4) que les deux occurrences généralisées représentent des occurrences provenant de la même séquence ($sid_1 = sid_2$), l'algorithme $cLocalTpJoin$ va construire une nouvelle occurrence généralisée (lignes 5 à 9).
Les lignes 5 et 6 attribuent respectivement les valeurs sid_1 et $tBeg_1$ aux champs sid et $tBeg$ de cette nouvelle occurrence.

La ligne 7 fixe la valeur du champs min en lui attribuant la valeur minimum de $[min_2, max_2]$ qui est supérieure ou égale à $min_1 + gapMin$. Cette valeur correspond à la date de fin de la première occurrence représentée par $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2 \rangle$ qui satisfait la contrainte de gap minimum avec le suffixe de la première occurrence représentée par $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1 \rangle$. Par exemple, sur la figure 3.20, cette occurrence correspondra à l'occurrence se

terminant sur le deuxième C de $[min_2, max_2]$.

De la même manière, c'est-à-dire dans un esprit de maximisation de l'intervalle de fin, la ligne 8 fixe la valeur de la borne supérieure de $[min,max]$, max, à la valeur maximum de $[min_2, max_2]$ qui est inférieure ou égale à $tBeg_1 + winMax$. Cette valeur correspond à la date de fin au plus tard d'une occurrence représentée par $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ qui vérifie la contrainte de *fenêtre de temps maximum* par rapport à $tBeg_1$. Par exemple, sur la figure 3.20, cette occurrence correspondra à l'occurrence se terminant sur le troisième C, en partant de la fin, de $[min_2, max_2]$.

Enfin, la ligne 9 fixe la valeur de $gmax$ pour la nouvelle occurrence. Celle-ci sera fixée à la valeur maximum de $[min_2, max_2]$ qui est inférieure ou égale à $gmax_1 + gapMax$. La valeur $gmax_1$ correspond à la date de fin maximum autorisée aux occurrences représentées par $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ pour satisfaire la contrainte de *gap maximum*. Cela signifie que $gmax$ indique la date de fin maximum des occurrences représentées par $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ qui peut former une occurrence de v vérifiant $gapMax$.

Cette fonction est appelée par la fonction $cTpJoin$ (Algorithme 11) qui génère une nouvelle liste d'occurrences généralisées contraintes à partir des listes de deux motifs générateurs z_1 et z_2.
En effet, dans cet algorithme (algorithme 11), les boucles imbriquées des lignes 1 et 2 permettent de considérer toutes les occurrences de $occList(z_1)$ et toutes celles de $occList(z_2)$. Pour chaque paire d'occurrences (occ_1,occ_2), une nouvelle occurrence généralisée contrainte est générée, quand cela est possible par un appel à la fonction $cLocalTpJoin$.

L'algorithme 11 est un algorithme de jointure général appelé lorsque les motifs générés sont des k-motifs tels que $k > 2$. Une jointure particulière est appliquée pour les 2-motifs. Cette procédure est appelée $cTpJoinLevel_2$ et est présentée dans l'algorithme 13.

L'algorithme 13 ($cTpJoinLevel_2$) est spécifique à la génération des listes d'occurrences généralisées contraintes des 2-motifs, c'est-à-dire lorsque les motifs générateurs z_1 et z_2 sont de simples items. Comme l'algorithme 11, cet algorithme génère une nouvelle liste d'occurrences à partir des listes de z_1 et de z_2. Il procède de la même manière, c'est-à-dire en considérant des paires d'occurrences construites à partir des deux listes d'entrée, ($\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ et $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$).

Contrairement à l'algorithme général $cTpJoin$ (algorithme 11), l'algorithme $cTpJoinLevel_2$ (algorithme 13) effectue plusieurs appels à la fonction de jointure locale $cLocalTpJoin$(algorithme 12). En effet, comme on a déjà pu l'aborder dans la présentation du champs $tBeg$, les occurrences des 1-motifs représentées par une occurrence généralisée doivent être considérées individuellement puisqu'elles cor-

Algorithme 13 ($cTpJoinLevel_2$)
Entrée : $occList(z_1)$, $occList(z_2)$
Fonction appelée : *cLocalTpJoin (Algorithme 12)*
Sortie : *une nouvelle liste d'occurrences généralisées contraintes, occList*

Initialiser occList à la liste vide.
1. **for all** $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle \in occList(z_1)$ **do**
2. **for all** $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle \in occList(z_2)$ **do**
3. **for all** $p \in [min_1 , max_1]$ **do**
4. **let** $\langle v, add\rangle := cLocalTpJoin(\langle sid_1, p, [p, max_1], p\rangle,$
$\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle)$
5. **if** *add* **then**
6. *Insert v in occList*
7. **fi**
8. **od**
9. **od**
10. **od**
11. **output** *occList*

FIG. 3.21 – Algorithme de jointure temporelle spécifique ($cTpJoinLevel_2$).

respondent à différentes dates de début possibles pour des occurrences du 2-motif traité. Ainsi, plusieurs appels sont effectués sur toutes les occurrences de la forme $\langle sid_1, p, [p, p], p\rangle$ et en faisant varier p entre min_1 et max_1.

La jointure équivalente contrainte :

L'opération de jointure équivalente contrainte est appelée pour construire la liste d'occurrences généralisées contraintes d'un motif dont les suffixes de ses motifs générateurs appartiennent à un même événement. Celle-ci procède de la même manière que la jointure équivalente non contrainte (algorithme 9) présentée dans la section 3.2, c'est-à-dire qu'elle considère toutes les paires d'occurrences des listes des motifs générateurs et fait appel à la fonction locale *cLocalEqJoin* (algorithme 15) afin de générer une nouvelle occurrence généralisée. Cette jointure locale (algorithme 10), commence par vérifier que les deux occurrences généralisées fournies en entrée de la fonction représentent bien des occurrences provenant d'une même séquence (ligne 3). L'algorithme s'assure ensuite que l'intersection entre les deux intervalles $[min_1, max_1]$ et $[min_2, max_2]$ n'est pas vide (ligne 4), ce qui signifie qu'il existe bien des occurrences représentées par $\langle sid_1, tBeg_1, [min_1, max_1],$

Algorithme 14 (*cEqJoin*)
Entrée : $occList(z_1)$, $occList(z_2)$
Fonction appelée : *cLocalEqJoin (algorithme 15)*
Sortie : *une nouvelle liste d'occurrences généralisées contraintes, occList*

1. **for all** $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle \in occList(z_1)$ **do**
2. **for all** $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle \in occList(z_2)$ **do**
3. **let** $\langle v, add\rangle := cLocalEqJoin(\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle, \langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle)$
4. **if** *add* **then**
5. *Insert v in occList*
6. **fi**
7. **od**
8. **od**
9. **output** *occList*

FIG. 3.22 – Algorithme de jointure équivalente contrainte (*cEqJoin*).

$gmax_1\rangle$ et par $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$ dont les suffixes apparaissent au même instant. Si tel est le cas, il construit alors une nouvelle occurrence dont l'intervalle de fin [*min*,*max*] est égal au résultat de l'intersection (lignes 6 et 7), et dont les valeurs de *tBeg* et de *gmax* sont respectivement égales à la date de début au plus tard entre $tBeg_1$ et $tBeg_2$ (ligne 5), et à la date au plus tôt entre $gmax_1$ et $gmax_2$ (ligne 8).

3.3.3 Correction de l'algorithme *GoSpec*

Dans l'approche par liste d'occurrences, le comptage du support d'un motif s'effectue en déterminant le nombre de *sid* distincts présents dans sa liste d'occurrences. Pour les listes d'occurrences généralisées contraintes, le procédé reste le même mais exclut du comptage les occurrences généralisées dont la valeur de *gmax* est -1, car celles-ci représentent des occurrences ne satisfaisant pas $gapMax$.

Afin de prouver la correction du comptage, nous devons montrer la justesse et la complétude des listes d'occurrences généralisées contraintes construites par *GoSpec*. La propriété de la justesse signifie dans ce cas que toute occurrence généralisée d'une liste, telle que $gmax \neq -1$, représente au moins une occurrence réelle vérifiant les contraintes. La propriété de complétude, quant à elle, signifie que pour toute occurrence réelle d'un motif vérifiant les contraintes, il existe une occurrence généralisée (avec $gmax \neq -1$) dans la liste du motif qui la représente.

Comme pour les démonstrations de *GoSpade* nous utilisons des notations

Algorithme 15 (*cLocalEqJoin*)
Entrée : *Deux occurrences généralisées contraintes* $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ *et*
$\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$
Sortie : $\langle v, add\rangle$, *où* $v = \langle sid, tBeg, [min, max], gmax\rangle$ *et add, une valeur booléenne égale à faux si v ne peut pas être créée.*

1. **let** $add := false$
2. **let** $v := null$
3. **if** $(sid_1 = sid_2)$ **then**
4. **if** $(min_1 \leq max_2$ *and* $max_1 \geq min_2)$ **then**
5. **let** $sid := sid_1$
6. **let** $tBeg := max(tBeg_1, tBeg_2)$
6. **let** $min := max(min_1, min_2)$
7. **let** $max := min(max_1, max_2)$
7. **let** $gmax := min(gmax_1, gmax_2)$
8. **let** $v := \langle sid, tBeg, [min, max], gmax\rangle$
9. **let** $add := true$
10. **fi**
11. **fi**
12. **output** $\langle v, add\rangle$

FIG. 3.23 – Algorithme de jointure équivalente locale contrainte (*cLocalEqJoin*).

allégées que nous rappelons ci-dessous : Une occurrence sera notée $y = e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{n-1} \rightarrow e_n$ au lieu de $y = e_1(eid(e_1)) \rightarrow e_2(eid(e_2)) \rightarrow \ldots \rightarrow e_{n-1}(eid(e_{n-1})) \rightarrow e_n(eid(e_n))$. Lorsque cela sera nécessaire, l'identifiant d'un événement e_i sera noté $t(e_i)$. Le préfixe d'une occurrence y, noté *préfixe*(y), représentera $e_1 \rightarrow e_2 \rightarrow \ldots \rightarrow e_{n-1}$ et son suffixe (*suffixe(y)*) sera égale à e_n. Nous utiliserons aussi *begin*(y) (resp. *end*(y)) pour dénoter l'*eid* du premier (resp. dernier) événement d'une occurrence y.

Définition 13 ***(justesse des listes)*** Soit z un motif et $occList(z)$ sa liste d'occurrences généralisées contraintes. $occList(z)$ est *juste* si pour tout v de $occList(z)$, où v est de la forme $\langle sid, tBeg, [min, max], gmax\rangle$, avec $gmax \neq -1$, il existe t_f appartenant à $[min, gmax]$ et il existe une occurrence y de z dans une séquence S telle que $end(y) = t_f$ et qui satisfait les contraintes $gapMin$, $gapMax$ et $winMax$.

Théorème 4 *Pour tout motif z, la liste d'occurrences généralisées contraintes, occList(z), générée par GoSpec est juste.*

Nous donnons la preuve du théorème suivant, qui est plus fort que le théorème 4 et qui l'implique trivialement.

Celui-ci présente deux différences. La première concerne le fait que le théorème suivant ne montre pas seulement l'existence d'une occurrence réelle pour au moins une date dans [*min*,*gmax*], mais va montrer que pour toutes les dates de cet intervalle, il existe une occurrence réelle. De plus, ces occurrences proviennent d'une date de fin appartenant à tout l'intervalle [*min*,*max*] (non plus uniquement à l'intervalle [*min*,*gmax*]) et *gmax* n'est plus contraint à être différent de -1. La deuxième différence repose sur la date de début considérée pour l'occurrence. Dans le théorème suivant, le choix de la date de début de l'occurrence est restreint puisque celle-ci est fixée à *tBeg*.

Théorème 5 Soit z un motif séquentiel et $occList(z)$ sa liste d'occurrences généralisées contraintes. $occList(z)$ est *juste* si pour tout v de $occList(z)$, où v est de la forme $\langle sid, tBeg, [min, max], gmax\rangle$, il existe une séquence S dans la base telle que $Id(S) = sid$ et pour tout entier t_f dans [*min*,*max*], il existe une occurrence y de z dans S telle que $begin(y) = tBeg$, $end(y) = t_f$, y satisfait $gapMin$, $winMax$, *préfixe(y)* satisfait $gapMax$ et si $t_f \in [min,gmax]$ alors y satisfait $gapMax$.

Preuve :
Nous ne considérons pas ici les cas de jointure équivalente (algorithme *cEqJoin* page 89), le principe étant similaire à la démonstration du théorème 1 de justesse de *GoSpade*.

La preuve de ce théorème pour tous les k-motifs sera faite par induction sur k.

Considérons $k > 2$. On suppose que les $occLists$ de tous les (k-1)-motifs sont *justes*.

Soit z un k-motif. Soit $v = \langle sid, tBeg, [min, max], gmax\rangle$ une occurrence généralisée contrainte provenant de $occList(z)$.
Nous allons montrer que $\exists S \in \beta$ telle que $sid = id(S)$ et telle que pour tout $t_{end} \in [min, max]$, il existe une occurrence y du motif z dans S de la forme $e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e_n$ telle que :

$$\begin{cases} begin(y) = tBeg \\ end(y) = t_{end} \\ y \text{ vérifie } gapMin \text{ et } winMax \\ \textit{préfixe(y)} \text{ vérifie } gapMax \\ y \text{ vérifie } gapMax \text{ si } end(y) \in [min, gmax] \end{cases}$$

Soit $v_1 = \langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ et $v_2 = \langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$, les deux occurrences généralisées contraintes à l'origine de la génération de v dans l'algorithme 12. Par construction, nous avons $sid = sid_1 =$

sid_2 et l'hypothèse de justesse au niveau k-1 nous permet d'affirmer qu'il existe $S \in \beta$ telle que $sid = id(S)$.

Considérons maintenant un t_{end} quelconque dans l'intervalle $[min,max]$. Par construction de min et max, $min \in [min_2, max_2]$ et $max \in [min_2, max_2]$. Ainsi, $t_{end} \in [min_2, max_2]$.

Avec l'hypothèse de justesse du niveau k-1 et puisque $t_{end} \in [min_2, max_2]$, nous pouvons affirmer qu'il existe $y_2 = e''_1 \rightarrow \ldots \rightarrow e''_{n-1}$, une occurrence de v_2 dans S telle que $t(e''_{n-1}) = t_{end}$.

Tout d'abord, considérons le cas où $t_{end} \in]gmax,max]$ (ce cas inclus les situations où $gmax = -1$).
La justesse du niveau k-1 permet aussi d'affirmer qu'il existe $y_1 = e'_1 \rightarrow \ldots \rightarrow e'_{n-1}$, une occurrence de v_1 qui est telle que $t(e'_{n-1}) = min_1$, $t(e'_1) = tBeg_1$, y_1 satisfait $winMax$ et $gapMin$, et puisque $t(e'_{n-1}) = min_1 \leq gmax_1$ (algorithme 12, ligne 3), y_1 satisfait $gapMax$.

Soit $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-1} \rightarrow e''_{n-1}$.

(Etape 1) Montrons que, $t(e''_{n-1}) - t(e'_1) \leq winMax$. Par construction (algorithme 12 ligne 8), $max \leq tBeg_1 + winMax$. Puisque $t_{end} \in [min,max]$, et $t(e'_1) = tBeg = tBeg_1$ (algorithme 12 ligne 6), on a $t(e''_{n-1}) = t_{end} \leq max \leq t(e'_1) + winMax$, c'est-à-dire que $t(e''_{n-1}) - t(e'_1) \leq winMax$.

(Etape 2) Montrons maintenant que y vérifie la contrainte de *gap minimum*. Par construction (algorithme 12 ligne 7), $min \geq min_1 + gapMin$, et puisque $min_1 = t(e'_{n-1})$ et $t(e''_{n-1}) = t_{end} \geq min$, alors $t_{end} - min_1 \geq min - min_1 \geq gapMin$, c'est-à-dire que $t(e''_{n-1}) - t(e'_{n-1}) \geq gapMin$. De plus, la propriété de justesse étant vérifiée pour y_1, on sait que $\forall i = 2, \ldots, n-1, t(e'_i) - t(e'_{i-1}) \geq gapMin$, ce qui termine de montrer que y vérifie la contrainte de *gap minimum*.

Enfin, puisque y_1 satisfait $maxGap$, alors *préfixe(y)* satisfait aussi $maxGap$. Nous avons ainsi montré que pour tout $t_{end} \in]gmax,max]$, il existe $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-1} \rightarrow e''_{n-1}$, une occurrence de z dans S telle que :

$$\begin{cases} t(e'_1) = tBeg \\ t(e''_{n-1}) = t_{end} \\ y \text{ vérifie } gapMin \text{ et } winMax \\ \textit{préfixe(y)} \text{ vérifie } gapMax \end{cases}$$

Considérons maintenant le second cas où $t_{end} \in [min,gmax]$.
Par construction (algorithme 12 lignes 7 et 9), $min \geq min_1 + gapMin$ et $gmax \leq gmax_1 + gapMax$. Puisque $t_{end} \geq min$ et $t_{end} \leq gmax$, alors $t_{end} - min_1 \geq gapMin$ et $t_{end} - gmax_1 \leq gapMax$. Considérons maintenant les trois situations suivantes : (i) $t_{end} - min_1 \leq gapMax$; (ii) $t_{end} - min_1 > gapMax$ et $t_{end} - gmax_1 \geq gapMin$; et (iii) $t_{end} - min_1 > gapMax$ et $t_{end} - gmax_1 < gapMin$.
En considérant la première situation, c'est-à-dire la situation (i) où $t_{end} - min_1 \leq gapMax$. La justesse du niveau k-1 nous permet de considérer le même y_1 que

précédemment. Soit $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-1} \rightarrow e''_{n-1}$. En appliquant le même raisonnement que précédemment (Etape 1, Etape 2), nous pouvons montrer que y est une occurrence de z dans S, qui est telle que :

$$\begin{cases} t(e'_1) = tBeg \\ t(e''_{n-1}) = t_{end} \\ y \text{ vérifie } gapMin \text{ et } winMax \\ \textit{préfixe(y)} \text{ vérifie } gapMax \end{cases}$$

Puisque $t_{end} - min_1 \leq gapMax$, c'est-à-dire que $t(e''_{n-1}) - t(e'_{n-1}) \leq gapMax$, et *préfixe(y)* satisfait $maxGap$, alors y satisfait aussi $maxGap$.

En considérant la deuxième situation (*ii*), nous avons $t_{end} - min_1 > gapMax$ et $t_{end} - gmax_1 \geq gapMin$. La justesse du niveau k-1 implique qu'il existe $y_1 = e'_1 \rightarrow \ldots \rightarrow e'_{n-1}$ une occurrence de v_1 telle que $t(e'_1) = tBeg_1$, $t(e'_{n-1}) = gmax_1$, et vérifiant $minGap$, $maxWin$ et $maxGap$ (puisque $t(e'_{n-1}) \in [min_1, gmax_1]$).

Soit $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-1} \rightarrow e''_{n-1}$. Le même raisonnement que précédemment (Etape 1) peut alors être appliqué pour montrer que $t(e''_{n-1}) - t(e'_1) \leq winMax$. Par construction de y_2 on a $t(e''_{n-1}) = t_{end}$ et par construction de y_1 on a $t(e'_{n-1}) = gmax_1$. Puisque nous considérons la situation où $t_{end} - gmax_1 \geq gapMin$, alors $t(e''_{n-1}) - t(e'_{n-1}) \geq gapMin$. Donc, comme y_1 satisfait $minGap$, nous avons y satisfait $minGap$. Par construction (algorithme 12 ligne 9), nous avons aussi $gmax \leq gmax_1 + gapMax$ et puisque $t_{end} \leq gmax$ alors $t_{end} - gmax_1 \leq maxGap$, et ainsi $t(e''_{n-1}) - t(e'_{n-1}) \leq gapMax$. Comme y_1 satisfait $maxGap$, alors nous avons aussi y satisfait $maxGap$. Nous avons ainsi montré que $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-1} \rightarrow e''_{n-1}$ est une occurrence de z dans S vérifiant :

$$\begin{cases} t(e'_1) = t_{beg} \\ t(e''_{n-1}) = t_{end} \\ y \text{ vérifie } gapMin, winMax \text{ et } gapMax \end{cases}$$

Considérons maintenant la dernière situation, c'est-à-dire la situation (*iii*) où $t_{end} - min_1 > gapMax$ et $t_{end} - gmax_1 < gapMin$. Montrons que $t_{end} - gapMax \in [min_1, gmax_1]$.

Puisque $gapMax \geq gapMin$ (sinon aucun motif ne peut être extrait), alors $t_{end} - gapMax \leq t_{end} - gapMin < gmax_1$. Nous avons donc, $t_{end} - gapMax \in [min_1, gmax_1]$.

La justesse du niveau k-1 implique qu'il existe $y_1 = e'_1 \rightarrow \ldots \rightarrow e'_{n-1}$ une occurrence de v_1 qui est telle que $t(e'_1) = tBeg_1$, $t(e'_{n-1}) = t_{end} - gapMax$, et y_1 satisfait $minGap$, $maxWin$ et $maxGap$.

Soit $y = e'_1 \rightarrow \ldots \rightarrow e'_{n-1} \rightarrow e''_{n-1}$. Le même raisonnement que précédemment (Etape 1) peut être appliqué pour montrer que $t(e''_{n-1}) - t(e'_1) \leq winMax$, c'est-à-dire que y satisfait $maxWin$. Puisque $t(e'_{n-1}) = t_{end} - gapMax$ et $t_{end} - gapMax \leq t_{end} - gapMin$, nous avons $t(e'_{n-1}) - t(e''_{n-1}) \geq minGap$. Comme y_1 satisfait

minGap, ceci implique que y vérifie *minGap*. Enfin, par choix de $t(e'_{n-1})$, nous avons $t_{end} - t(e'_{n-1}) = gapMax$ et puisque $t_{end} = t(e''_{n-1})$, alors $t(e''_{n-1}) - t(e'_{n-1}) = gapMax$. Comme y_1 satisfait *maxGap*, alors y le satisfait aussi.

Nous avons ainsi montré que y est une occurrence de z dans S telle que :

$$\begin{cases} t(e'_1) = t_{beg} \\ t(e''_{n-1}) = t_{end} \\ y \text{ vérifie } gapMin, winMax \text{ et } gapMax \end{cases}$$

Pour compléter cette preuve, considérons les cas où $k = 1$ et $k = 2$.

Soit $k = 1$. Par construction, les *occLists* des items sont justes.

Soit $k = 2$. Le même raisonnement appliqué pour l'algorithme 12 peut être appliqué à l'algorithme $cTpJoinLevel_2$ pour montrer que les *occLists* des 2-motifs sont justes.

◇

Pour la démonstration de complétude, nous introduisons la définition préliminaire suivante.

Définition 14 ***(v représente y)*** Soit y une occurrence d'un motif z dans une séquence S provenant d'une base de séquences β. Soit $occList(z)$ la liste d'occurrences généralisées contraintes de ce motif et v un élément de $occList(z)$, représentée par le tuple $\langle sid, tBeg, [min, max], gmax \rangle$. On dit que v *représente* y si $sid(v) = Id(S)$, $begin(y) \leq tBeg$, $min \leq end(y) \leq max$ et si y satisfait la contrainte $gapMax$ alors $end(y) \leq gmax$.

Définition 15 ***(complétude de la représentation)*** Soit z un motif séquentiel et $occList(z)$ sa liste d'occurrences généralisées contraintes. $occList(z)$ est *complète* si pour tout S dans β et pour tout y occurrence de z dans S satisfaisant $gapMin$, $winMax$ et $gapMax$, il existe v dans $occList(z)$ qui *représente* y.

Théorème 6 *Pour tout motif z, la liste d'occurrences généralisées contraintes, occList(z), générée par GoSpec est complète.*

Pour les mêmes raisons que pour le théorème 4, nous donnons la preuve du théorème suivant, qui implique trivialement le théorème 6.

Théorème 7 Soit z un motif séquentiel et $occList(z)$ sa liste d'occurrences généralisées contraintes. $occList(z)$ est *complète* si pour tout S de β et pour tout occurrence y du motif z dans S satisfaisant $gapMin$ et $winMax$ et qui est telle que *préfixe(y)* satisfait $gapMax$, alors il existe v dans $occList(z)$ qui *représente* y.

Preuve :

Comme pour la démonstration de justesse, nous ne considérons pas ici les cas de jointure équivalente (algorithme *cEqJoin* page 89), le principe étant similaire à la démonstration du théorème 2 de complétude de *GoSpade*.

La preuve pour les k-motifs est effectuée par induction sur k.

Considérons k tel que $k > 2$, et supposons que les listes d'occurrences généralisées contraintes (*occLists*) sont complètes pour tous les (k-1)-motifs.

Soit z un k-motif, S une séquence de β et soit y une occurrence du motif z telle que y vérifie $gapMin$ et $winMax$, et telle que *préfixe(y)* vérifie $gapMax$. Nous voulons montrer qu'il existe $v \in occList(z)$ telle que v représente y. Le motif z est de la forme $p \rightarrow s_1 \rightarrow s_2$, où p est un $(k-2)$-motif et s_1, s_2 sont de simples items.

Puisque z est de la forme $p \rightarrow s_1 \rightarrow s_2$, l'occurrence y de z est de la forme $e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_{n-1} \rightarrow e_n$ où $e_{n-1} = \{s_1\}$ et $e_n = \{s_2\}$.

Soit $y_1 = e_1 \rightarrow \ldots \rightarrow e_{n-1}$ et $y_2 = e_1 \rightarrow \ldots \rightarrow e_{n-2} \rightarrow e_n$. Ainsi, y_1 (resp. y_2) est une occurrence d'un (k-1)-motif z_1 (resp. z_2) dans S, vérifiant $minGap$ et $maxWin$, et telle que *préfixe*(y_1) (resp. *préfixe*(y_2)) satisfait $maxGap$.

L'hypothèse de complétude du niveau k-1 nous permet d'affirmer qu'il existe v_1 dans $occList(z_1)$ (resp. il existe v_2 dans $occList(z_2)$) telle que y_1 (resp. y_2) est représentée par v_1 (resp. v_2). Soit v_1 et v_2 de la forme respective $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle$ et $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle$.

Nous montrons que durant la construction de $occList(z)$, obtenue par un appel à $TpJoin$ entre $occList(z_1)$ et $occList(z_2)$, une occurrence généralisée contrainte v a été générée à partir de v_1 et de v_2 et qu'elle est telle que v représente y.

L'occurrence y vérifie la contrainte de *gap minimum*, c'est-à-dire que $\forall i = 2, \ldots, n, t(e_i) - t(e_{i-1}) \geq gapMin$, et en particulier pour $i = n$, $t(e_n) - t(e_{n-1}) \geq gapMin$. De plus, puisque $t(e_{n-1}) \in [min_1, max_1]$ (car v_1 représente y_1), alors $t(e_{n-1}) \geq min_1$. Ainsi, $t(e_n) \geq min_1 + gapMin$. Enfin, puisque v_2 représente y_2, alors $t(e_n) \in [min_2, max_2]$, c'est-à-dire que $t(e_n) \leq max_2$, et nous avons $max_2 \geq min_1 + gapMin$. Ainsi, la première condition de la conjonction de la ligne 3 de l'algorithme 12 est satisfaite.

Nous avons $t(e_n) - t(e_1) \leq winMax$, puisque y vérifie la contrainte de *fenêtre de temps maximum*, et $t(e_n) \geq min_2$ (v_2 représente y_2), alors $t(e_1) + winMax \geq min_2$. Puisque $t(e_1) \leq tBeg_1$ (v_1 représente y_1), $tBeg_1 + winMax \geq min_2$ et ainsi la seconde condition de la ligne 3 de l'algorithme 12 est satisfaite.

Comme *préfixe(y)*, c'est-à-dire y_1, vérifie $maxGap$ alors, puisque v_1 représente y_1, nous avons $t(e_{n-1}) \in [min_1, max_1]$ avec $t(e_{n-1}) \leq gmax_1$. D'où $min_1 \leq gmax_1$, ce qui signifie que la dernière condition de la ligne 3 de l'algorithme 12 est aussi satisfaite.

L'hypothèse de complétude du niveau k-1 implique que $sid_1 = Id(S)$ et que

sid_2=$Id(S)$, c'est-à-dire que $sid_1 = sid_2$. Ainsi, l'exécution atteint la ligne 5 de l'algorithme 12. Montrons maintenant qu'une occurrence généralisée contrainte $v = \langle sid, tBeg, [min, max], gmax \rangle$ est générée, et qu'elle est telle que v représente y (i.e., $sid = Id(S)$, $t(e_1) \leq tBeg$, $min \leq t(e_n) \leq max$ et $end(y) \leq gmax$ si y satisfait $gapMax$).

Par construction de sid et $tBeg$ (Algorithme 12 lignes 5 et 6), et puisque v_1 représente y_1, nous avons $sid = Id(S)$ et $t(e_1) \leq tBeg$. Maintenant, nous allons montrer que min et max existent et vérifient $min \leq t(e_n) \leq max$. Puisque v_1 représente y_1, alors $t(e_{n-1}) \geq min_1$, et puisque $t(e_n) - t(e_{n-1}) \geq gapMin$ (car y vérifie la contrainte de *gap minimum*), nous avons $t(e_n) \geq min_1 + gapMin$. Ainsi, il existe $x \in [min_2, max_2]$ (e.g. $x = t(e_n)$) tel que $x \leq t(e_n)$ et $x \geq min_1 + gapMin$, ce qui signifie que min (algorithme 12 ligne 7) existe et $min \leq t(e_n)$.

Puisque $t(e_n) \in [min_2, max_2]$ (v_2 représente y_2) et $t(e_n) \leq tBeg + winMax$ (y satisfait $winMax$), nous savons que max (algorithme 12 ligne 8), l'élément maximum de $[min_2, max_2]$ tel que $max \leq tBeg + winMax$ existe et $t(e_n) \leq max$. Puisque $min \leq t(e_n) \leq max$, nous avons aussi trivialement $min \leq max$.

Puisque min et max existent et $min \leq max$, l'exécution atteint la ligne 12 de l'algorithme 12 et l'occurrence $v = \langle sid, tBeg, [min, max], gmax \rangle$ sera générée.

Nous avons montré que $sid = Id(S)$, $t(e_1) \leq tBeg$ et $min \leq t(e_n) \leq max$, donc, pour montrer que v représente y, il reste encore à établir que si y satisfait $gapMax$, nous avons aussi $end(y) \leq gmax$ (i.e., $t(e_n) \leq gmax$). Supposons que y satisfait $gapMax$, alors $t(e_n) - t(e_{n-1}) \leq gapMax$. Puisque *préfixe(y)* satisfait $gapMax$, alors y_1 satisfait $gapMax$ et ainsi, la complétude du niveau k-1 garantie que $t(e_{n-1}) \leq gmax_1$. D'où, $t(e_n) \leq gmax_1 + gapMax$. Puisque $t(e_n) \in [min_2, max_2]$ (v_2 représente y_2), et que $gmax$ est l'élément x maximum appartenant à $[min_2, max_2]$ et vérifiant $x \leq gmax_1 + gapMax$ (algorithme 12, ligne 9), nous avons alors $t(e_n) \leq gmax$.

Pour compléter la preuve par induction sur la taille des motifs, nous devons considérer les cas où $k = 1$ et $k = 2$.

Soit $k = 1$. Par construction, les *occLists* des items sont complètes.

Soit $k = 2$. Le même raisonnement que pour l'algorithme 12 peut être appliqué pour l'algorithme $cTpJoinLevel_2$ afin d'établir que les *occLists* des 2-motifs sont complètes.

◇

L'algorithme *GoSpec* détermine le support d'un motif candidat z en comptant le nombre de sid distincts apparaissant dans $occList(z)$ dès lors que $gmax \neq -1$. Les théorèmes 4 et 6 assurent la correction de cette procédure de comptage.

3.3.4 Résultats expérimentaux

Nous présentons des résultats expérimentaux qui montrent que les extractions de motifs séquentiels généralisés basées sur une représentation par listes d'occurrences sont grandement améliorées par l'utilisation d'occurrences généralisées contraintes lorsque les jeux de données contiennent des répétitions consécutives. Dans ces expériences, nous comparons les comportements de *GoSpec* et de *cSpade* [Zak00], la version contrainte de l'algorithme *Spade* [Zak01, Zak98].

Les algorithmes *GoSpec* et *cSpade* ont été développés en utilisant Microsoft Visual C++ 6.0, avec le même niveau d'optimisation afin de faciliter leur comparaison. Toutes les expériences ont été exécutées sur un PC avec 196 MB de mémoire et un processeur Pentium III 500MHz sous Microsoft Windows 2000.

Les expériences ont été exécutées à partir du jeu de données synthétique (set_1) généré en utilisant le générateur DataQuest d'IBM[2] [AS94b] et présenté dans la section 3.2.4. Ce jeu contient 1000 séquences construites à partir d'un alphabet de 100 items et présentent un longueur moyenne de 10 événements par séquence. Dans ce jeu de données, l'intervalle de temps entre deux événements est fixé à 1 et il y a un seul événement par unité de temps.

Nous rappelons ici le post-traitement appliqué à ce jeu afin d'obtenir plusieurs jeux de données contenant des répétitions consécutives. Chaque item rencontré dans un événement d'une séquence a une probabilité fixée à 10% d'être répété. Lorsqu'un item est répété, nous le répétons simplement dans les i événements consécutifs. Si la fin d'une séquence est atteinte durant le processus de duplication, nous n'ajoutons pas de nouvel événement à la séquence et l'item courant n'est alors pas complètement dupliqué. Nous notons $set1\_r\{i\}$ le jeu obtenu avec un paramètre de répétition fixé à i. Dans un soucis d'uniformité, le jeu initial $set1$ est noté $set1\_r0$. Le post-traitement de $set1\_r0$ amène à la création de 5 nouveaux jeux $set1\_r1$,..., $set1\_r5$. Tous ces jeux ont été créés simultanément afin que ce soit les même items qui soit répétés dans les différents jeux. Par exemple, si l'item A est un item à répéter et qu'il apparaît dans une séquence à l'instant 5, alors il sera ajouté à l'événement se trouvant à l'instant 6 dans le jeu $set1\_r1$, aux événements se trouvant aux instants 6 et 7 dans le jeu $set1\_r2$, et ainsi de suite.

Les graphiques des figures 3.24, 3.25 et 3.26 montrent les résultats d'extractions effectuées sur les jeux de données $set1\_r1$, $set1\_r2$, ..., $set1\_r5$ en utilisant les contraintes suivantes : un seuil de fréquence minimum fixé à 2.5%, une contrainte de fenêtre de temps maximum fixée à 6, un gap minimum fixé à 2 et un gap maximum fixé à 4.

Le graphique de la figure 3.24 donne la taille des listes d'occurrences (en terme de nombre d'éléments) de *cSpade* et de *GoSpec* pour des extractions effectuées sur les jeux de données $set1\_r1$, ..., $set5\_r5$. Comme nous pouvions l'imaginer, le

[2] *http ://www.almaden.ibm.com/cs/quest/syndata.html*

nombre d'occurrences utilisées par *cSpade* est supérieur au nombre d'occurrences généralisées utilisées par *GoSpec*. De plus, cette diminution augmente avec le nombre de répétitions.

Le graphique de la figure 3.25 montre que cette réduction a un impacte direct sur le coût des opérations de *jointure* (en terme de nombre d'appels à $LocalJoin$), qui résulte en une réduction importante du temps total d'exécution des extractions, comme le montre le graphique de la figure 3.26.

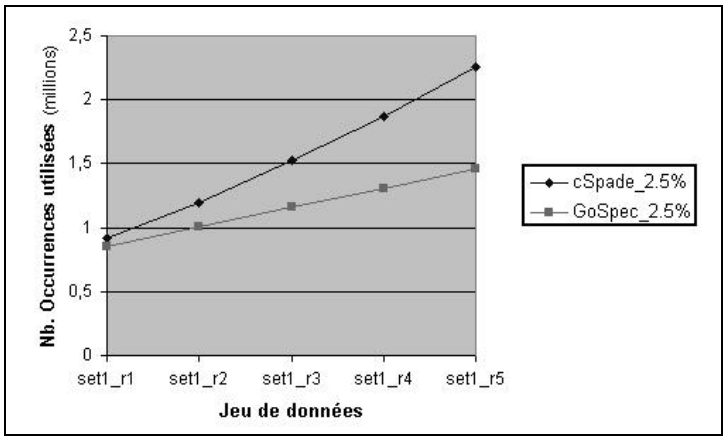


FIG. 3.24 – Evolution du nombre total d'occurrences utilisées.

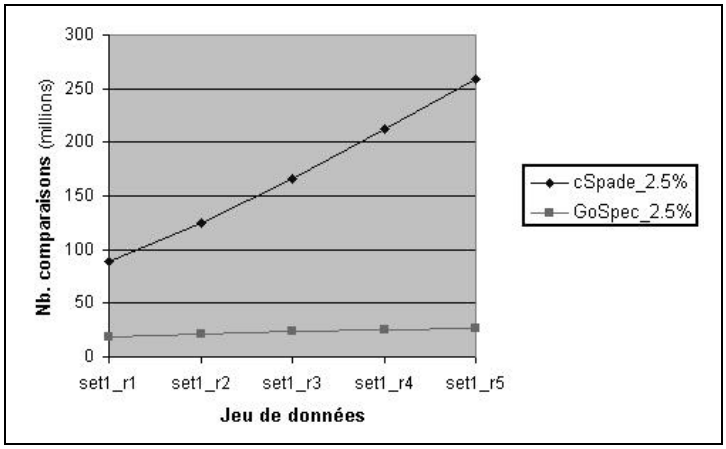


FIG. 3.25 – Evolution du nombre total de comparaisons effectuées.

Enfin, le graphique de la figure 3.27 complète ces résultats avec les temps d'exécution pour des extractions effectuées sur les jeux $set1\_r0$ et $set1\_r5$. Il montre

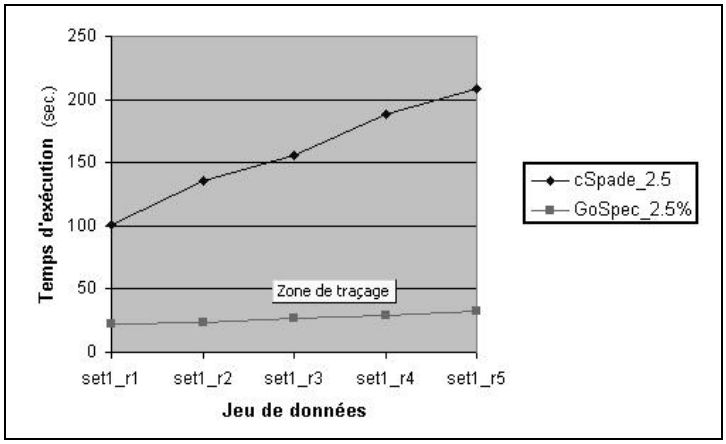


FIG. 3.26 – Evolution du temps d'exécution.

que le temps d'exécution nécessaire à *GoSpec* pour extraire un certain nombre de motifs ne varie que très peu en présence de répétitions consécutives.

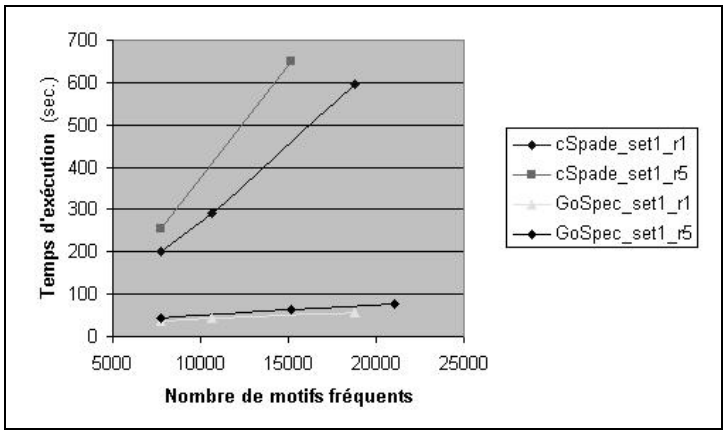


FIG. 3.27 – Influence des répétitions consécutives sur *cSpade* et *GoSpec*.

3.4 Bilan

Dans ce chapitre nous avons montré les limites des algorithmes basés sur une approche par listes d'occurrences lorsque les données contiennent des répétitions consécutives. La solution apportée repose sur une représentation plus générale des occurrences des motifs qui permet de condenser plusieurs occurrences en une seule

occurrence généralisée, réduisant ainsi la taille des listes d'occurrences. Nous avons présenté les algorithmes dédiés à la gestion de cette nouvelle représentation dans le cas d'extractions de motifs séquentiels fréquents (algorithme *GoSpade*) et dans le cas d'extractions contraintes (algorithme *GoSpec*). Pour chacun de ces algorithmes, nous avons effectué une démonstration de justesse et de complétude assurant la correction des résultats qu'ils retournent, et montré par des expérimentations effectuées sur des jeux de données réelles et synthétiques que ces algorithmes améliorent nettement les performances de deux algorithmes représentatifs de cette famille (*Spade* et *cSpade*).

Dans le chapitre suivant, nous abordons les aspects pratiques de ces algorithmes en présentant les techniques d'implémentations utilisées ainsi que les optimisations apportées lors du développement de *GoSpade* et de *GoSpec*. Dans la deuxième partie du chapitre, nous décrivons une application réelle effectuée dans le cadre industriel de cette thèse effectuée à la Caisse des Dépôts et Consignations et qui concerne le traitement de situations boursières représentées par des séquences d'événements financiers.

Chapitre 4

GoSpade et *GoSpec*, en pratique

Le passage d'algorithmes abstraits aux traitements de données réelles ne peut être efficace si le développement des algorithmes s'effectue de façon naïve. C'est pourquoi, avant de présenter une application réelle effectuée au sein de la Caisse des Dépôts et Consignations, nous présentons certaines techniques d'implémentation efficaces des algorithmes *GoSpade* et *GoSpec*. Ces techniques apportent des optimisations intéressantes aux niveaux de la consommation mémoire et du coût de jointure, qui représentent les limites principales des algorithmes de la troisième famille à laquelle nos algorithmes appartiennent.

4.1 Points d'implémentations

4.1.1 Gestion de la mémoire

Une approche par listes d'occurrences nécessite de garder en mémoire les listes d'occurrences des motifs générés. L'espace mémoire utilisé est alors important et peut parfois dépasser largement les capacités possédées par un ordinateur. Afin de limiter cette consommation, il est possible d'agir sur plusieurs points. Par exemple, l'algorithme *Spade* propose de ne construire les listes d'occurrences qu'à partir du niveau 2, c'est-à-dire pour les 2-motifs fréquents. L'économie, en terme de nombre d'occurrences à stocker, est importante puisque c'est à ce niveau que les listes d'occurrences sont les plus longues. Cette première optimisation peut s'accompagner d'autres optimisations qui permettent d'une part de libérer la mémoire au plus tôt et d'autre part de réduire les allocations nécessaires à un processus d'extraction.

Algorithme 16 (*GoSpade* optimisé)
Entrée : *Une base de séquences β, un support minimum minSup et un motif p représentant le préfixe courant.*
Sortie : *Pas de valeur de retour, mais écriture des fréquents dans un fichier de sortie f*
Premier appel : *GoSpade(β,minSup,$\emptyset$)*

1. **if** *($p = \emptyset$)* **then**
2. *Construire $GoIdList[i]$ pour tout $i \in I$*
3. *$F_{[\emptyset]} := \{i \in I | support(i) \geq minSup$ dans $\beta\}$*
4. *écrire les éléments de $F_{[\emptyset]}$ dans f*
5. **fi**
6. **for all** $z_i \in F_{[p]}$ **do**
7. **for all** $z_j \in F_{[p]}$, *tel que* $z_j \geq z_i$ **do**
8. **for all** *z obtenu par* $fusion(z_i, z_j)$ **do**
9. $GoIdList[z] := jointure(GoIdList[z_i], GoIdList[z_j])$
10. *Utilise $GoIdList[z]$ pour déterminer si z est fréquent*
11. **if** *z est fréquent* **then**
12. *écrire z dans f*
13. $F_{[préfixe(z)]} := F_{[préfixe(z)]} \cup \{z\}$
14. **else**
15. *effacer $GoIdList[z]$*
16. **fi**
17. **od**
18. **od**
19. *effacer $GoIdList[z_i]$*
20. *GoSpade(β,minSup,z_i)*
21. **od**
22.effacer $F_{[p]}$

FIG. 4.1 – Algorithme général *GoSpade* optimisé.

Minimisation du stockage mémoire

Pour illustrer le principe, considérons l'algorithme *GoSpade* optimisé (algorithme 16). Notons que ces optimisations sont valables pour *GoSpec* puisqu'il ne diffère de *GoSpade* que par les type de jointure appliquées.

Afin de gérer l'espace mémoire efficacement, nous proposons de ne plus stocker l'ensemble des motifs fréquents dans une variable globale ($Freq$), mais de les écrire à la volée dans un fichier de sortie dès lors qu'ils sont identifiés comme fréquents (ligne 12). Il reste néanmoins nécessaire de conserver ces motifs provisoirement, puisqu'ils participent à la génération d'autres motifs. Pour cela, nous proposons d'utiliser deux dictionnaires associatifs, notés F et $GoIdList$. Une élément $F[p]$, du dictionnaire associatif F, permet d'associer à un motif p, un ensemble de motifs fréquents ayant pour préfixe p et un élément $GoIdList[p]$ du dictionnaire associatif $GoIdList$ permet d'associer à un motif p, un ensemble d'occurrences.

Ces deux dictionnaires sont gérés de façon globale durant l'exécution de l'algorithme *GoSpade* optimisé. Cependant, à la différence de la variable globale $Freq$ utilisée initialement, les informations conservées dans ces dictionnaires associatifs ont une durée de vie qui se limite aux seuls besoins du processus d'extraction. Ainsi, l'information contenue dans le dictionnaire associatif F, concernant un motif z (accédée par F[z]), peut être supprimée dès lors que tous les motifs ayant pour préfixe z ont été générés (ligne 22). De même, il n'est pas nécessaire de conserver les listes d'occurrences des motifs non fréquents (ligne 15), ni même de conserver celles des motifs qui ne seront plus utilisés comme motif générateur (ligne 19).

La gestion de la mémoire ainsi faite permet de libérer les espaces utilisés, au plus tôt, réduisant ainsi l'espace total consommé lors d'un processus d'extraction. Elle peut aussi agir, de façon indirecte, sur le temps global d'exécution, en évitant au maximum le recours au mécanisme de *swap mémoire*.

Réutilisation de la mémoire

Le bénéfice apporté par une libération au plus tôt de l'espace mémoire concerne le volume utilisé à un instant donné par le processus d'extraction. Maintenant, un autre coût important lié à une exécution concerne les allocations répétées d'espace mémoire. Il est possible d'agir sur ce point en réutilisant l'espace libéré, sans avoir à effectuer une nouvelle allocation.

Par exemple, lorsque les occurrences d'un motif z, stockées dans $GoIdList[z]$, ne sont plus nécessaires, il est possible de réutiliser directement, entièrement ou en partie, l'espace mémoire correspondant pour stocker les informations concernant un motif z', et notamment ses occurrences générées lors de l'opération de jointure (ligne 9, algorithme 16).

Algorithme 17 (*cTpJoinLevel*$_2$ optimisé)
Entrée : *occList*(z_1), *occList*(z_2)
Fonction appelée : *Algorithme cLocalTpJoin (algorithme 12)*
Sortie : *Une nouvelle liste d'occurrences généralisées contraintes, occList*

Initialiser occList à la liste vide.
1. **for all** $\langle sid_1, tBeg_1, [min_1, max_1], gmax_1\rangle \in occList(z_1)$ **do**
2. **for all** $\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle \in occList(z_2)$ **do**
3. **let** $q :=$ *l'élément maximum* $x \in [min_1, max_1]$ *tq.* $x \leq min_2 - gapMin$
4. **let** $r := minimum(max_1, max_2 - 1)$
5. **for all** $p \in [q,r]$ **do**
6. **let** $\langle v, add\rangle := cLocalTpJoin(\langle sid_1, p, [p, max_1], p\rangle,$
$\langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle)$
7. **if** *add* **then**
8. *Insère v dans occList*
9. **fi**
10. **od**
11. **od**
12. **od**
13. **output** *occList*

FIG. 4.2 – Algorithme *cTpJoinLevel*$_2$ optimisé.

4.1.2 Optimisation de la génération contrainte des 2-motifs

L'algorithme $cTpJoinLevel_2$ optimisé (algorithme 17) correspond à une version optimisée de l'algorithme de jointure spécifique, $cTpJoinLevel_2$, appelé lors de la construction des listes d'occurrences des motifs de taille 2 (algorithme 13 page 88).

Les optimisations apportées permettent de réduire le nombre d'appels (algorithme 17 ligne 6) effectués à la procédure *cLocalTpJoin*. Ces appels se limitent maintenant aux valeurs couvertes par un intervalle $[q, r] \in [min_1, max_1]$ (ligne 5), et non plus sur l'intervalle $[min_1, max_1]$ complet, comme c'était le cas initialement (algorithme 13 ligne 3). Les bornes q et r se justifient de la façon suivante.

Tout d'abord, la borne minimum de ce nouvel intervalle (valeur q calculée ligne 3), représente la valeur maximale de $[min_1, max_1]$ qui vérifie la contrainte du *gap minimum* avec la première occurrence contenue dans $[min_2, max_2]$. Ce choix ne modifie pas les occurrences générées par $cTpJoinLevel_2$. Ainsi, considérons l'appel $cLocalTpJoin(\langle sid_1, p, [p, max_1], p\rangle, \langle sid_2, tBeg_2, [min_2, max_2], gmax_2\rangle)$, avec $p = q$. la valeur *min* retournée par l'exécution de la ligne 7 de l'algorithme 12 (*cLocalTpJoin*), correspond à la valeur minimale de $[min_2, max_2]$qui vérifie la contrainte de *gap minimum*, c'est-à-dire min_2. Si on considère une valeur q' dans $[min_1, q[$, celle-ci donnera la même valeur de *min* puisque $q' < q$, ce qui signifie que les appels à *cLocalTpJoin* avec des valeurs de p inférieures à q sont inutiles.

La valeur de la borne maximum, r (ligne 4), permet quant à elle, de limiter le dépliage des *tBeg* aux seules dates de début qui peuvent représenter des occurrences valides. En effet, si on considère un $r' > r$, alors on aura $r' > max_2 - 1$, i.e., $r' \geq max_2$. Donc, aucune occurrence commençant à la date r' ne pourra se terminer à une date de l'intervalle $[min_2, max_2]$.

Ainsi, seules les dates de début appartenant à l'intervalle $[q,r]$ peuvent générer des occurrences généralisées nouvelles et valides. Il est donc possible de faire l'économie des jointures concernant les autres dates, puisque celles-ci n'apporteront pas d'informations nouvelles.

Cette optimisation permet de réduire le nombre d'appels effectués à l'algorithme *cLocalTpJoin* (algorithme 12 page 85), i.e., le nombre de comparaisons effectuées durant une opération de *jointure*.

4.1.3 Optimisation des opérations de *jointure*

Un autre coût important inhérent à une approche par listes d'occurrences concerne le coût des opérations de jointure. En effet, celles-ci ont une complexité en $\mathcal{O}(mn)$ où m (resp. n) représente le nombre d'occurrences composant la liste du premier (resp. deuxième) motif générateur. Les temps de calculs peuvent alors devenir très importants. Afin de diminuer les coûts des ces opérations, plusieurs optimisations peuvent être apportées aux différents algorithmes.

Travail à partir de listes d'occurrences triées

Afin de réduire le nombre de couples d'occurrences à considérer lors d'une opération de jointure, il est intéressant de travailler sur des listes triées dans l'ordre croissant. Ainsi, l'algorithme pourra travailler sur des paquets d'occurrences ayant la même valeur de sid, évitant ainsi d'avoir à traiter toutes les paires d'occurrences possibles. La complexité des opérations de jointure passera alors en $\mathcal{O}(\sum m_i n_i)$ avec m_i (resp. n_i) égal au nombre d'occurrences de la liste du premier (resp. deuxième) motif générateur représentant des occurrences appartenant à la $i^{ème}$ séquence. De même, l'utilisation de listes triées facilite le repérage des doublons, comme nous en verrons l'usage dans la section suivante.

En pratique, le coût associé au tri d'une liste est important. Il est alors intéressant d'éviter d'avoir à trier les listes afin d'économiser sur ce point. C'est pourquoi, nous avons veillé, lors de la génération des listes d'occurrences, à produire des listes déjà ordonnées selon le champs sid (et $tBeg$ dans le cas de $GoSpec$).

Dédoublonnage des listes d'occurrences contraintes

La construction des listes d'occurrences généralisées peut entraîner la construction d'occurrences généralisées contenant des informations redondantes.

Soient $occGen_1 = \langle sid_1, tBeg_1, [min_1, max_1], gmax_1 \rangle$ et $occGen_2 = \langle sid_2, tBeg_2, [min_2, max_2], gmax_2 \rangle$, deux occurrences généralisées contraintes, telles que $sid_1 = sid_2$, obtenues lors de la construction d'une liste par appel à l'algorithme $cTpJoin$ (algorithme 11 présenté page 84). Nous avons identifié les cas suivants, pour lesquels il est possible de ne conserver qu'une seule occurrence généralisée sur les deux.
(les exemples illustrant ces différents cas proviennent de la base de la figure 3.16 page 79).

- cas 1 : $occGen_1 = occGen_2$.

  Il est clair que dans ce cas on ne conserve qu'une seule de ces deux occurrences généralisées contraintes puisqu'elles couvrent exactement les mêmes occurrences.

- cas 2 : $occGen_1$ et $occGen_2$ sont telles que $tBeg_1 = tBeg_2$, $min_1 < min_2$, $max_1 = max_2$ et $gmax_1 = gmax_2$.

  Cette situation signifie que la deuxième occurrence généralisée contrainte représente moins d'occurrences que la première, puisqu'elle couvre les occurrences se terminant uniquement à partir de min_2. Dans ce cas, seule la première occurrence généralisée contrainte est conservée.

- cas 3 : $occGen_1$ et $occGen_2$ sont telles que $tBeg_1 < tBeg_2$, $min_1 = min_2$, $max_1 \leq max_2$ et $gmax_1 \leq gmax_2$.

  Pour simplifier, considérons tout d'abord la situation particulière où $tBeg_1 < tBeg_2$ avec égalité sur les autres champs. Cette situation signifie que les occurrences généralisées représentées par $occGen_1$ sont aussi représentées par $occGen_2$ (cf. définition 14 de **v représente y** page 94), ce qui permet de ne conserver que la deuxième occurrence $occGen_2$, c'est-à-dire l'occurrence possédant la date de début au plus tard, sans perte d'information.
  C'est par exemple la cas pour les occurrences généralisées ⟨1, 4, [9, 12], 12⟩ et ⟨1, 5, [9, 12], 12⟩ obtenues après avoir effectué une jointure entre les listes des motifs A → B et A → C. En effet, ⟨1, 4, [9, 12], 12⟩ représente des occurrences se terminant aux mêmes dates de fin que ⟨1, 5, [9, 12], 12⟩, il n'est donc pas nécessaire de conserver la première occurrence. La liste du motif A → B → C représentée sur la figure 3.16 (page 79) intègre déjà ce dédoublonnage.

  Maintenant, ce choix reste valable si $max_1 < max_2$ et $gmax_1 < gmax_2$ puisque dans ce cas $occGen_2$ représente plus d'occurrences que $occGen_1$.

  Cette situation se présente aussi de la construction de la liste d'occurrences du motif A → C, puisque le résultat de l'appel à la jointure temporelle spécifique (algorithme 13, page 88) devrait contenir, entre autre, les deux occurrences suivantes : ⟨2, 2, [5, 12], 7⟩ et ⟨2, 3, [5, 13], 8⟩. Dans ce cas, seule l'occurrence généralisée ⟨2, 3, [5, 13], 8⟩ est conservée puisqu'elle couvre plus d'occurrences que la première. La liste de ce motif, représentée sur la figure 3.16, page 79, intègre aussi ce dédoublonnage.

- cas 4 : $occGen_1$ et $occGen_2$ sont telles que $tBeg_1 = tBeg_2$, $min_1 = min_2$, $max_1 < max_2$ et $gmax_1 \leq gmax_2$.

  Dans ce cas, seule l'occurrence généralisée contrainte couvrant le maximum d'occurrences est conservée, c'est-à-dire $occGen_2$. Cette situation peut notamment se produire lors d'une opération de jointure pour des motifs de taille $k > 2$. Considérons par exemple la liste d'occurrences généralisées du motif A → C → B, que l'on obtiendrait à partir des listes d'occurrences des motifs A → C et A → B. La jointure entre ces deux listes pour les occurrences généralisées contraintes de la première séquence considère, en omettant les couples dont les occurrences généralisées ne proviennent pas de la même séquence, les couples d'occurrences suivants : (⟨1, 5, [9, 12], 10⟩,⟨1, 3, [6, 13], 8⟩), (⟨1, 5, [9, 12], 10⟩,⟨1, 4, [6, 14], 9⟩) et (⟨1, 5, [9, 12], 10⟩,⟨1, 5, [7, 15], 10⟩). Ces couples d'occurrences donnent lieu à la génération de 3 nouvelles occurrences généralisées, respectivement : ⟨1, 5, [11, 13], 13⟩, ⟨1, 5,

[11, 14], 14⟩ et ⟨1, 5, [11, 15], 15⟩. Dans ce cas, seule l'occurrence généralisée contrainte ⟨1, 5, [11, 15], 15⟩ est conservée puisqu'elle couvre un total de 5 occurrences contre 3 et 4 pour les deux autres. Ce cas est illustré sur la figure 4.4 page 110.

Optimisation de la jointure contrainte

La suppression des occurrences généralisées contraintes redondantes, telles que celles décrites dans la section précédente, permet de réduire le coût associé aux opérations de jointure puisque le nombre de couples d'occurrences à considérer est plus faible.

Dans cette section nous allons montrer que le nombre de couple considérés peut être encore considérablement réduit. Pour cela, étudions l'algorithme de jointure temporelle locale, *cLocalTpJoin* (algorithme 12), présenté dans la section 3.3. On remarque que celui-ci ne fait appel ni au champs $tBeg_2$, ni au champs $gmax_2$, ce qui signifie que la liste d'occurrences du deuxième motif générateur peut en fait être ramenée à une liste d'occurrences non contrainte.

La transformation d'une liste d'occurrences généralisées contraintes en une liste d'occurrences généralisées non contraintes peut être effectuée très rapidement. Cela consiste d'une part à projeter la liste sur les champs *sid*, *min* et *max* afin de passer à une liste d'occurrences non contraintes, et ensuite à procéder à un dédoublonnage de la façon suivante : si deux occurrences généralisées $occGen_1$ et $occGen_2$ sont telles que $[min_1, max_1] \cap [min_2, max_2] \neq \emptyset$, alors ces deux occurrences généralisées sont ramenées à une seule occurrence généralisée dont l'intervalle de fin correspond à l'union de ces intervalles ($[min_1, max_1] \cup [min_2, max_2]$). Ce dédoublonnage est beaucoup plus efficace s'il est effectué sur des listes triées aussi sur *min* et *max*, puisque dans ce cas, le dédoublonnage peut s'effectuer localement, c'est-à-dire en ne considérant que des paires d'occurrences successives. On rappelle que, par construction, les listes se trouvent déjà pré-triées sur les deux premiers champs, ce qui permet de réduire le coût associé au tri. De plus, dans la pratique, le gain apporté par cette optimisation couvre largement le coût nécessaire au tri des listes.

Par exemple, les trois occurrences généralisées contraintes, représentant des occurrences du motif A → B dans la première séquence de la base de la figure 3.16, sont ramenées à une seule occurrence généralisée non contrainte comme illustré sur la figure 4.3 : Tout d'abord, la projection des trois occurrences généralisées contraintes ⟨1, 3, [6, 13], 8⟩, ⟨1, 4, [6, 14], 9⟩ et ⟨1, 5, [7, 15], 10⟩, donne trois occurrences généralisées non contraintes ⟨1, [6, 13]⟩, ⟨1, [6, 14]⟩ et ⟨1, [7, 15]⟩. Ensuite, le dédoublonnage de proximité permet de ramener ces trois occurrences à une seule occurrence ⟨1, [6, 15]⟩, dont l'intervalle de fin correspond à l'union des trois intervalles de fin (([6 13] ∪ [6 14]) ∪ [7 15]).

La taille de la deuxième liste utilisée par *cLocalTpJoin* est considérablement

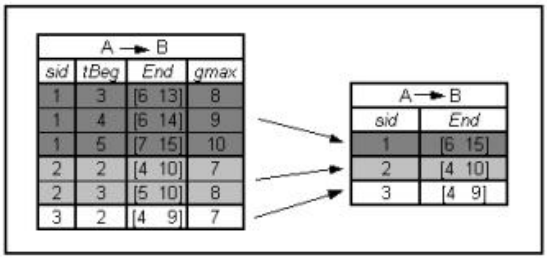


FIG. 4.3 – Passage de listes contraintes (*occList*) à des listes non contraintes (*GoIdList*).

réduite par cette modification, ce qui permet de réduire le nombre de couples d'occurrences à considérer lors d'une opération de jointure. Ce point est illustré sur la figure 4.4 qui représente la jointure temporelle (optimisée et non optimisée) effectuée entre les listes d'occurrences du motif A → C et A → B afin de construire la liste d'occurrences généralisées contraintes du motif A → C → B. Initialement, l'opération de jointure entre ces deux listes aurait considéré 6 couples d'occurrences (liens entre les deux premières listes du haut). Avec cette optimisation, celle-ci n'en considère plus que 3 (liens entre les deux premières listes du bas) : ($\langle$1, 5, [9, 12], 10$\rangle$,$\langle$1, [6, 15]$\rangle$), ($\langle$2, 3, [5, 13], 8$\rangle$,$\langle$2, [4, 10]$\rangle$) et ($\langle$3, 2, [8, 12], $-1\rangle$,$\langle$3, [4, 9]$\rangle$). On notera que cette dernière paire d'occurrences ne génère pas de nouvelle occurrence puisqu'elle ne satisfait pas le test $min_1 \leq gmax_1$ (ligne 3 de l'algorithme 12 page 85).

Au final, les résultats obtenus, avec et sans optimisation, sont identiques. En effet, l'opération de dédoublonnage présentée dans la section précédente, s'applique à la liste du motif A → C → B obtenue sans optimisation, et permet de ne conserver que deux occurrences parmi les 5 générées (cas 4 pour les occurrences de la première séquence et cas 1 pour les occurrences de la deuxième séquence).

4.1.4 Gains expérimentaux apportés par ces optimisations

Dans cette section, nous illustrons le gain obtenu, en pratique, par ces différentes optimisations. Les résultats présentés concernent des extractions effectuées avec l'algorithme *GoSpec*(section 3.3), sur un jeu composé de 574 séquences. Chaque séquence comporte en moyenne 15 événements construits à partir d'un alphabet de taille 20. Toutes les expérimentations ont été effectuées en appliquant une contrainte de *fenêtre de temps maximum* fixée à 10, ainsi qu'un *gap minimum* de 2 et un *gap maximum* de 4.

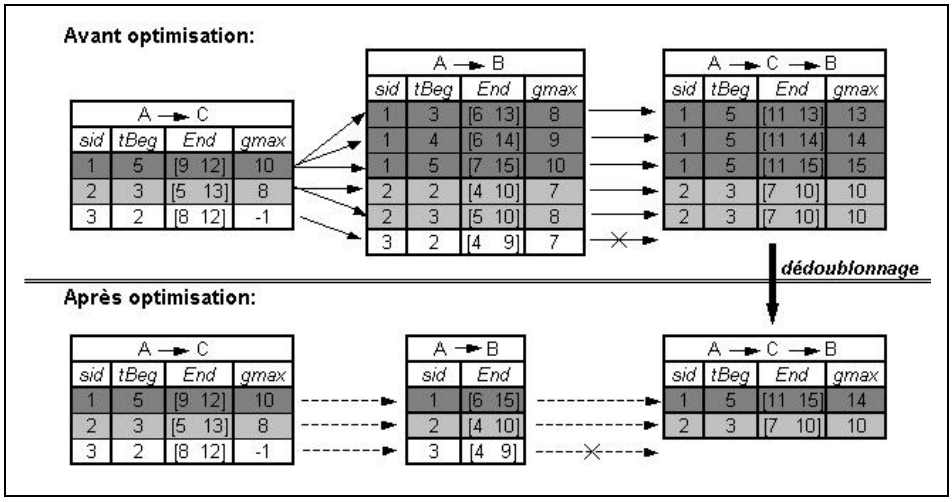


FIG. 4.4 – Effet des optimisations.

Réduction de la taille des listes

La figure 4.5 présente les gains apportés, en terme de nombre d'occurrences générées, par le dédoublonnage des listes d'occurrences. Les courbes représentent l'évolution de la longueur moyenne des listes d'occurrences créées par *GoSpec* et *GoSpec optimisé*, en fonction du seuil de support minimum (variant entre 5% et 75%).

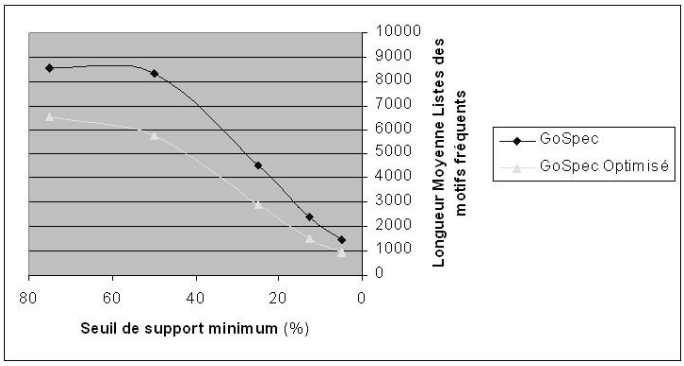


FIG. 4.5 – Gain obtenu sur la taille des listes d'occurrences.

Le gain le plus important est obtenu pour des seuils de fréquences élevés puisque les optimisations portent alors sur des listes dont la longueur moyenne est plus grande que celle des motifs peu fréquents.

Réduction du nombre de comparaisons

Le gain obtenu, en terme de nombre de comparaisons effectuées, est illustré par la figure 4.6. Celui-ci découle d'une part du dédoublonnage des listes d'occurrences et d'autre part des optimisations apportées aux jointures (jointure spécifique des 2-motifs et jointures effectuées sur une projection des listes des deuxièmes motifs générateurs).

Ces courbes montrent nettement que le gain obtenu est d'autant plus important que les extractions sont effectuées à faible seuil. Ceci s'explique par le fait que le nombre de motifs générés, et donc le nombre de jointures à effectuer, est beaucoup plus important qu'à seuil élevé. De plus, il est intéressant de noter que la pente de la courbe de *GoSpec* optimisé est beaucoup plus faible que celle de *GoSpec*.

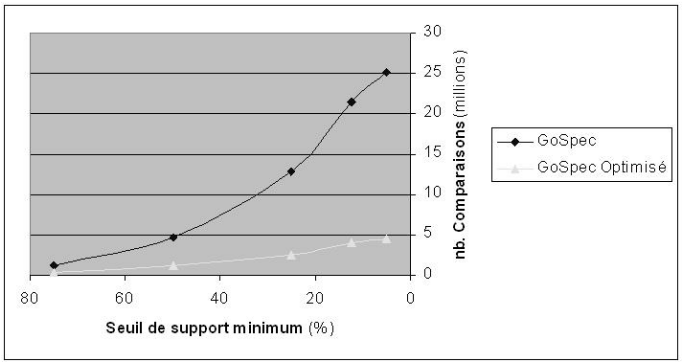


FIG. 4.6 – Gain obtenu sur le nombre de jointures effectuées.

Réduction du temps d'exécution

L'application des différences optimisations présentées dans cette section (réutilisation de la mémoire, dédoublonnage des listes, travail sur des listes d'occurrences projetées pour les deuxièmes motifs générateurs), permet de réduire de manière significative le temps d'exécution global. Ce gain est illustré par la figure 4.7.

Passage à l'échelle

La figure 4.8 montre que les optimisations apportées à l'algorithme *GoSpec* permettent de travailler sur des jeux de données volumineux et à des seuils faibles en un temps raisonnable. La courbe représente l'évolution du temps d'exécution de *GoSpec optimisé* en fonction de la taille des jeux (en terme de nombre de séquences). Les extractions ont été effectuées à un seuil de fréquence minimum de 15% avec une contrainte de *gap maximum* fixée à 4, sur des bases de séquences dont la taille varie de 20 000 à 100 000 séquences. Ces bases proviennent toutes d'un

même jeu de données synthétique comportant 100 000 séquences et obtenu en utilisant le générateur d'IBM DataQuest ([AS94b]) avec les paramètres de génération suivants : C10-T2.5-S4-I1.25-D100K et un alphabet de 1000 items.

Dans la section suivante, nous montrons une utilisation concrète de l'extracteur *GoSpec* optimisé, dans le cadre d'une application effectuée à la Caisse des Dépôts et Consignations sur des données financières.

4.2 Application au domaine de la finance

Les données biologiques et financières sont des exemples types de données contenant des répétitions consécutives. Leur présence dans des données biologiques, telles des bases de séquences d'ADN ou encore de séquences protéiques, s'explique par la faible taille des alphabets nucléiques (composés des 4 nucléotides Adénine, Thymine, Guanine et Cytosine) et protéiques (composés des 21 acides aminés), ce qui augmente leur probabilité d'apparition dans chaque événement. La présence de répétitions dans des séquences provenant de données financières s'explique par la transformation des données quantitatives, tels des cours boursiers, en événements financiers (cf. section 4.2.1). L'utilisation de l'algorithme *GoSpec* pour des applications biologiques ou financières se trouve alors justifiée par la présence de ces répétitions. Les figures 4.9 et 4.10 illustrent l'intérêt de *GoSpec* sur ce type de données.

La première figure (figure 4.9) concerne des résultats d'extractions effectuées sur une base de séquences d'ADN contenant 1778 séquences. Chaque séquence est composée de 102 événements en moyenne et chaque événement contient un seul item provenant de l'alphabet nucléique ({A,T,G,C}). Les courbes représentent l'évolution des temps d'exécution obtenus par *cSpade* et *GoSpec* pour des extractions effectuées avec une contrainte de *fenêtre de temps maximum* de 6 et un *gap maximum* fixé à 3, avec des seuils de support minimum variant entre 5% et 25%. Ces courbes montrent les avantages de *GoSpec*, en pratique, sur des données biologiques.

La figure 4.10, quant à elle, représente des résultats d'extractions obtenus par *cSpade* et *GoSpec*, sur un jeu de données qui concerne les variations d'un cours boursier sur une année. La base de séquences contient 2830 séquences ayant une longueur moyenne de 15 événements (c'est-à-dire couvrant une période de 15 minutes). Les séquences ont été construites sur un alphabet de 17 items. Cette figure montre que l'utilisation de *GoSpec* présente un gain significatif sur *cSpade*, ce qui prouve là encore l'intérêt de cet algorithme sur ce type de données.

Dans les sections suivantes, nous présentons une application qui concerne le domaine de la finance. Celle-ci entre dans le cadre professionnel de ma thèse CIFRE effectuée au sein de la Caisse des Dépôts et Consignations.

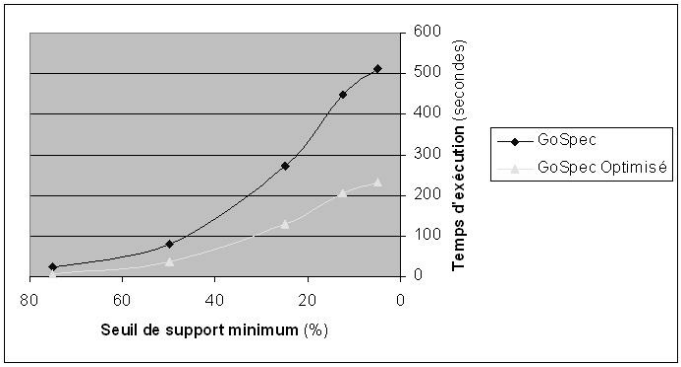


FIG. 4.7 – Gain obtenu sur le temps d'exécution.

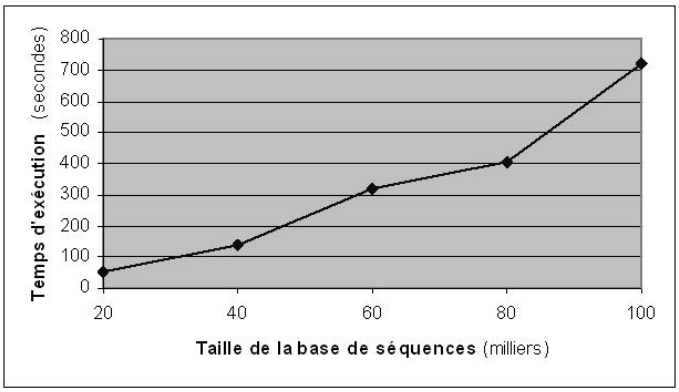


FIG. 4.8 – Evolution du temps d'exécution lorsque la taille des bases de séquences augmentent ($minSup$= 15%, $maxGap$= 4).

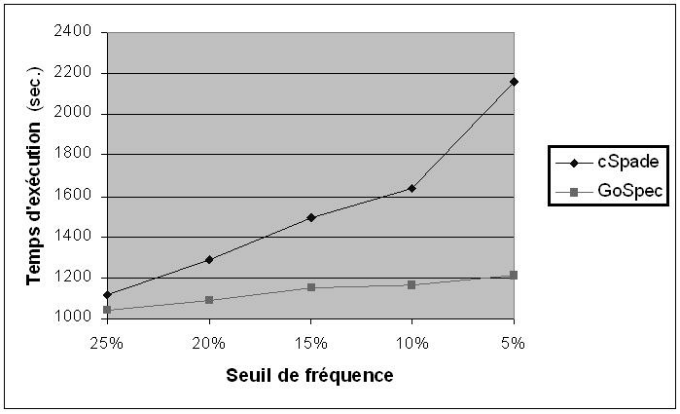


FIG. 4.9 – Comportement de *GoSpec* sur des données biologiques.

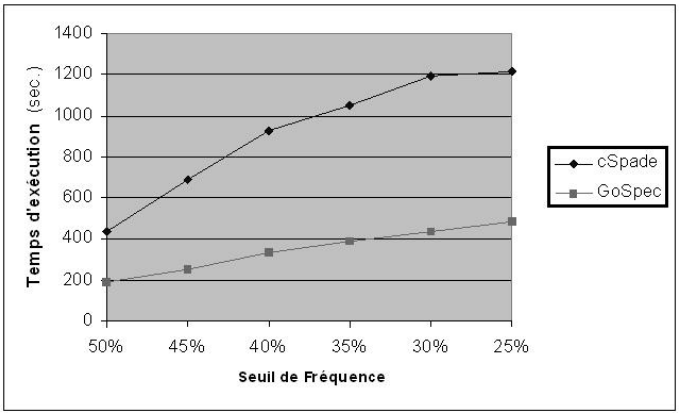


FIG. 4.10 – Comportement de *GoSpec* sur des données financières.

L'origine de cette application concerne une étude qui vise à obtenir des signatures de situations boursières, représentant différentes classes de tendances, et pouvant être présentées aux opérateurs de marché afin d'appuyer leur prise de décision.

L'approche utilisée consiste à signer ces différentes classes de situations par un ensemble de motifs séquentiels caractéristiques. La notion de motifs caractéristiques est une notion vaste. Par exemple, un motif caractéristique peut être un motif contenant un item ou un sous-motif particulier ([KPWD03]), un motif qui apparaît fréquemment dans un jeu et pas dans un autre ([DL99]), ou encore un motif ayant une fréquence comprise entre une valeur minimum et maximum ([DK01, KD01]). Cependant, pour cette application, nous avons choisi de limiter cette notion à un ensemble de motifs séquentiels fréquents vérifiant des combinaisons de contraintes temporelles (fenêtre de temps maximum et gap minimum et maximum). Nous avons pour cela travaillé sur des classes constituées de situations boursières représentées au préalable par des séquences d'événements financiers.

Dans un premier temps, nous présentons (section 4.2.1) le processus appliqué afin d'obtenir les motifs caractéristiques constituant les signatures de chaque classe. Nous présentons ensuite (section 4.2.2) des résultats expérimentaux qui montrent que les signatures ainsi constituées sont porteuses d'informations spécifiques à chacune des classes.

4.2.1 Signature de situations boursières

Dans cette section nous décrivons la technique employée pour obtenir une signature de situations boursières composée de motifs séquentiels caractéristiques.

Les différentes classes traitées sont au nombre de trois, et sont constituées d'un ensemble de dates auxquelles les experts financiers ont identifié des tendances à la hausse (classe *hausse*), à la baisse (classe *baisse*) ou sans changement significatif du marché (classe *stabilité*).

La figure 4.11 illustre le processus général de signature de ces différentes classes. Celui-ci comporte deux étapes principales : la première étape consistant à transformer les données en séquences d'événements financiers (section 4.2.1) et la deuxième correspondant à la recherche des motifs séquentiels caractéristiques (section 4.2.1).

Construction des séquences d'événements financiers

La démarche que nous proposons fait appel à des données quantitatives continues, que nous discrétisons, et à des données qualitatives (e.g., dépêches de presse financières). Pour des raisons des compatibilité des ensembles de données disponibles, nous n'avons utilisé que des données quantitatives.

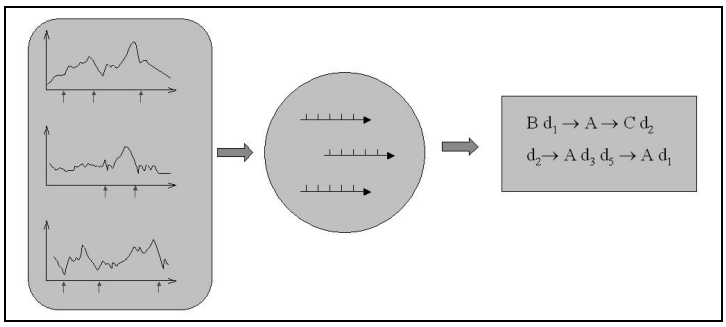


FIG. 4.11 – Signature de situations boursières : principe.

Celles-ci concernent l'évolution du cours *France Telecom* durant l'année 1999. Ces données ont été fournies par la *Caisse des Dépôts et Consignations*. Afin de composer nos séquences d'événements, nous avons considéré l'évolution de trois valeurs durant cette période. La première concerne le prix d'achat et la seconde le volume d'achat. La dernière valeur concerne un indicateur propre à la Caisse des Dépôts et Consignations dont nous ne pouvons donner la composition pour des raisons de confidentialité. Cet indicateur sera noté *IND*.

Au départ, les classes sont constituées par un ensemble de dates auxquelles les experts ont identifié une tendance particulière. A chaque date de ces trois classes (*Hausse*, *Baisse* et *Stabilité*), nous associons une séquence d'événements financiers qui représente l'évolution des trois indicateurs durant les 15 minutes précédentes. Ces séquences sont construites à partir de trois alphabets distincts associés respectivement aux *prix* de transactions ($\mathcal{A}_{prix}$), aux *volumes* de transactions ($\mathcal{A}_{volume}$) et à l'indice IND ($\mathcal{A}_{IND}$), chaque événement financier contient ainsi trois items (un item pour chaque valeur). Ces alphabets contiennent tous 9 items ({A,B,C,D,E,F,G,H,I}), et permettent d'indiquer d'une part le *sens de variation* de la valeur et d'autre part son *intensité de variation*.

Le principe appliqué est le suivant :
Soit $x(t)$ la courbe représentant la valeur concernée (*prix*, *volume* ou IND), $MME(t)$ sa moyenne mobile et $\sigma(t)$ son écart type mobile.

La moyenne mobile est un type de moyenne qui permet de lisser la courbe $x(t)$ en effectuant la moyenne des valeurs précédentes avec une pondération plus grande pour les valeurs les plus récentes. Une moyenne mobile (notée MME) d'une valeur (x) à l'instant t est calculée de la façon suivante :

$$MME(t) = MME(t-1) + \beta * (x(t) - MME(t-1))$$

Le facteur β représente le facteur de pondération appliqué sur les valeurs

précédentes (facteur d'oubli). Celui-ci est égal à $2/(n+1)$ où n représente la fenêtre de temps à laquelle on s'intéresse.

L'écart-type mobile représente l'écart moyen d'une valeur à sa moyenne mobile. Celui-ci se calcule de la façon suivante :

$$\sigma^2(t) = \sigma^2(t-1) + \beta * ((x(t) - MME(t))^2 - \sigma^2(t-1))$$

La figure 4.12 représente l'évolution d'une valeur en fonction du temps, sa moyenne mobile (courbe en gras) et l'intervalle de variation borné par l'écart-type mobile (courbes en pointillés).

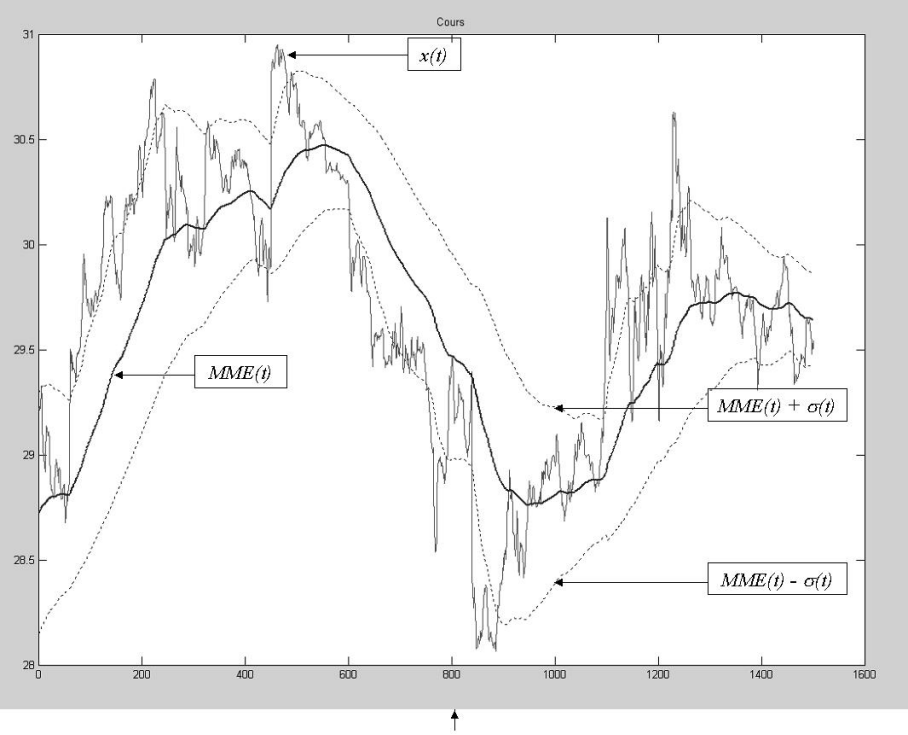


FIG. 4.12 – cours-MME-Sigma mobile.

A chaque instant T, on cherche à attribuer un item de l'alphabet à l'écart que présente la valeur $x(T)$ avec sa moyenne mobile $MME(T)$ (soit $écart(T) = x(T) - MME(T)$). L'attribution d'un item s'effectue en deux temps : dans un premier temps, le signe de cet écart nous permet de connaître le *sens de variation* de la valeur et dans un deuxième temps, la valeur de cet écart nous permet de savoir si l'écart est un écart normal ou non (i.e., l'*intensité de variation*). Pour cela, trois

limites ont été fixées afin de séparer ces intensités : γ_1, γ_2 et γ_3. Le tableau de la figure 4.13 représente l'item correspondant à un sens de variation et à l'intensité de variation de l'écart. La valeur de ces différents seuils est fixée de telle sorte que chaque item d'un alphabet apparaisse avec une fréquence significative.

écart (T)	Positif	Négatif	Nul
$\geq \gamma_1 \times \sigma(T)$	A	F	E
$\geq \gamma_2 \times \sigma(T)$	B	G	
$\geq \gamma_3 \times \sigma(T)$	C	H	
$< \gamma_3 \times \sigma(T)$	D	I	

FIG. 4.13 – Signification des alphabets.

Une fois les données initiales transformées en séquences d'événements financiers, nous avons obtenu trois jeux de données, représentant chacun une classe de tendance. Chaque jeu se compose d'un ensemble de séquences : 480 séquences pour la classe *Hausse*, 1143 séquences pour la classe *Baisse* et 1209 séquences pour la classe *Stabilité*.
Ces séquences sont composées en moyenne de 15 événements, soit 45 items puisque chaque événement contient 3 items.

Nous venons de voir de quelle façon les données quantitatives considérées pouvaient être transformées en séquences d'événements financiers. A ces données quantitatives discrétisées, nous proposons d'ajouter des données qualitatives, telles des dépêches de presse financières. Celles-ci n'ont pu être utilisées dans les expériences que nous présentons, pour des raisons de compatibilité des périodes couvertes par les ensembles dont nous disposions. Nous pouvons néanmoins décrire de quelles façon celles-ci auraient pu être intégrées aux séquences, dès lors qu'un ensemble de catégories de mots-clés, appelées *descripteurs*, ont été définis.

D'une façon générale, les séquences sont construites à partir d'un alphabet de descripteurs. Les descripteurs sont des étiquettes données à différents thèmes identifiés dans des dépêches de presse et sont constitués d'un ensemble de mots-clés qui ont une signification proche.

Par exemple, si on considère la dépêche de presse n_i, parue à une date j (figure 4.14) :

Rachat de l'entreprise américaine X par le géant pétrolier Y.

Dans cette dépêche, trois mots-clés ont pu être identifiés : *rachat*, *américaine* et *pétrolier*. Ces trois mots-clés appartiennent chacun à une catégorie couvrant un thème plus général représentée respectivement par les descripteurs *Rachat*, *USA* et *Energie*. Afin de faciliter l'encodage de la séquence, chaque descripteur est représenté par un item. Dans notre cas, l'item d_1 représente le descripteur *Rachat*, d_2 représente *USA* et d_3 le descripteur *Energie*.

Une fois qu'une dépêche a été réduite à un ensemble de descripteurs, celle-ci peut être encodée dans la séquence en formant un événement à la date j constitué des items correspondant à chaque descripteur. Ainsi, la dépêche de presse n_i a été encodée dans la séquence sous la forme d'un événement $\{d_1,d_2,d_3\}$ apparaissant à une date j.

Lorsqu'une application considère à la fois des données quantitatives et des données qualitatives, les séquences obtenues par les différents pré-traitements sont fusionnées afin de constituer une séquence composée d'items mixtes.

L'étape de pré-traitement terminée, nous avons des classes composées de séquences d'événements pour lesquelles nous allons rechercher les motifs séquentiels caractéristiques qui seront utilisés pour expliquer chaque classe. Comme nous l'avons précisé en début de section, les motifs constituant les différentes classes sont des motifs séquentiels. Nous allons maintenant décrire les différentes classes de séquences obtenues, ainsi que les contraintes appliquées pour extraire les motifs séquentiels souhaités.

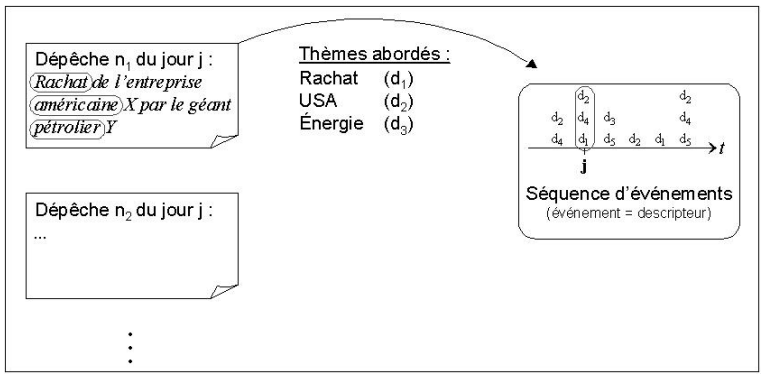


FIG. 4.14 – Exemple de pré-traitement de données qualitatives (dépêches de presse).

Extraction de motifs séquentiels caractéristiques

Les motifs qui vont constituer les signatures des différentes classes ont été extraits en faisant intervenir, en plus d'une contrainte de fréquence minimum fixée à 20%, des contraintes de temps et en particulier une contrainte de fenêtre de temps maximum et une contrainte de gap maximum. Le choix de ces contraintes se justifie de la façon suivante : on a en effet estimé qu'une tendance ne pourrait sûrement pas être expliquée par des motifs couvrant une période de temps trop longue et constitués d'événements trop éloignés dans le temps. Ainsi, la contrainte de fenêtre de temps a été fixée à 10 minutes, i.e., $winMax = 10$ (les séquences couvrant une période de 15 minutes), et la contrainte de gap maximum a été fixée à 4 minutes ($winMax = 4$).

Les extractions ont été effectuées sur chacune des trois bases de séquences obtenues précédemment (cf. section 4.2.1. Le nombre de motifs séquentiels extraits pour chaque classe est précisé dans le tableau de la figure 4.15. Ainsi, les extractions effectuées ont permis d'obtenir des classes composées respectivement de 1113 motifs séquentiels pour la classe *Hausse*, 1173 pour la classe *Baisse* et 1425 pour la classe *stabilité*.

Parmi ces motifs, certains sont communs aux trois classes. Afin que chaque classe de tendance ait une signature qui lui soit propre, nous avons supprimé ces motifs. La composition finale de ces signatures est présentée sur la dernière ligne du tableau de la figure 4.15.

	Hausse	*Baisse*	*Stabilité*
Nombre de séquences	480	1143	1209
Nombre de motifs extraits	1113	1173	1425
Nombre de motifs des signatures	**143**	**223**	**377**

FIG. 4.15 – Description des signatures obtenues.

Les signatures ainsi obtenues sont composées de motifs rares (puisque seuil faible), portant sur une période de temps raisonnable (fenêtre de temps maximum) et dont les événements ne sont pas trop éloignés ($maxGap$). Ces signatures ayant pour objectif de fournir de l'information aux opérateurs de marché, nous montrons, dans la section suivante, que ces signatures contiennent une information spécifique à la tendance qu'elles représentent.

4.2.2 Classification de situations boursières

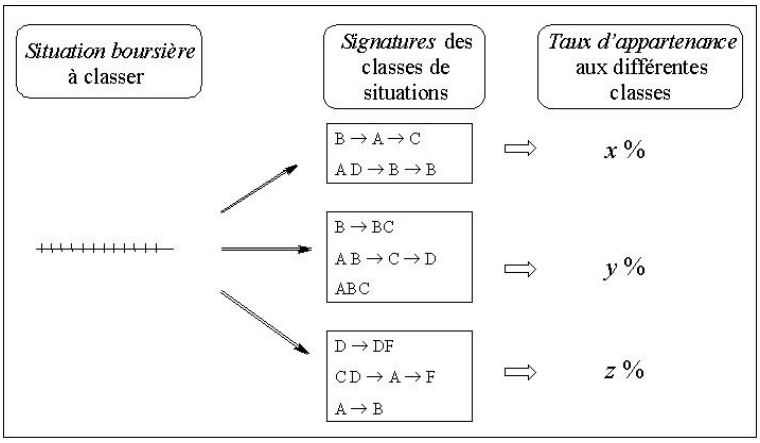


FIG. 4.16 – Principe de classification de situations boursières.

Afin de montrer que les signatures contiennent bien de l'information spécifique, nous avons classé de nouvelles situations boursières, dont la tendance était connue, en utilisant l'information contenue dans les différentes signatures. Le principe de classification d'une séquence test est illustré sur la figure 4.16. Tout d'abord, soit $\mathcal{S}_1$ la signature d'une classe $\mathcal{C}_1$ composée de n motifs caractéristiques (m_i, $i = 1, \ldots, n$). Le taux d'appartenance d'une séquence s test à $\mathcal{C}_1$ représente le pourcentage de motifs m_i contenus dans cette séquence. Le taux d'appartenance de cette séquence s est calculé de cette façon pour chacune des trois classes. Ensuite, la séquence est attribuée à une classe $\mathcal{C}$ dès lors que son taux d'appartenance est supérieur à celui qu'il obtient pour les deux autres classes. Ainsi, soit $x\%$ (resp. $y\%$ et $z\%$) le taux d'appartenance obtenu pour une séquence s dans la signature $\mathcal{S}_1$ (resp. $\mathcal{S}_2$ et $\mathcal{S}_3$) de la classe $\mathcal{C}_1$ (resp. $\mathcal{C}_2$ et $\mathcal{C}_3$). La séquence s sera attribuée à la classe $\mathcal{C}_1$ si $x > y$ et $x > z$.

La connaissance a priori de la tendance de chaque séquence test nous a permis de calculer le taux de séquences test bien classées. Une séquence est bien placée si elle a été attribuée à la classe correspondant à sa tendance. Les résultats sont présentés dans le tableau de la figure 4.17. Les cases grisées de la diagonale correspondent au pourcentage de séquences bien classées pour chaque tendance. Dans le cas d'une classification aléatoire, la probabilité de classer correctement une séquence est de 33,33%. Les scores obtenus pour une classification à partir des signatures de motifs séquentiels sont supérieurs à cette valeur : 50% pour la classe *Hausse*, 48% pour la classe *Baisse* et 40% pour la classe *Stabilité*. Ces résultats montrent que les signatures ainsi constituées contiennent une information

spécifique à la classe qu'elles représentent.

Classe estimée \ Classe réelle	*Hausse*	*Baisse*	*Stabilité*
Hausse	50 %	25.5 %	32.5 %
Baisse	23 %	48 %	27.5 %
Stabilité	27 %	26.5 %	40 %

FIG. 4.17 – Résultats de classification de situations boursières.

4.3 Conclusion

Dans ce chapitre, nous avons abordés les questions de mises en pratique des algorithmes *GoSpade* et *GoSpec* présentés dans le chapitre 3. Ainsi, dans une première partie nous avons présenté et développé les principaux points techniques utilisés pour effectuer une implémentation efficace de nos algorithmes. Les optimisations apportées interviennent sur les points les plus délicats des approches par listes d'occurrences, c'est-à-dire au niveau de la consommation de la mémoire et des coûts des opérations de jointure. Nous avons montré que les gains obtenus, en terme d'espace mémoire et de temps d'exécution, par ces prototypes étaient significativement intéressants comparés aux résultats obtenus lorsque les extractions étaient effectuées à partir d'une implémentation naïve des algorithmes.

La deuxième partie de ce chapitre présentait une application réelle effectuée à la Caisse des Dépôts et Consignations. Celle-ci concerne le traitement de données financières représentées par des séquences d'événements dont l'objectif est de caractériser différentes tendances de marché par des ensembles de motifs séquentiels fréquents. Nous avons pu montrer que les résultats obtenus contenaient une information bien spécifique à chaque tendance considérée.

Chapitre 5

Conclusions et Perspectives

Les travaux de recherche présentés dans ce mémoire concernent le domaine de la fouille de données (data mining) et plus particulièrement celui de l'extraction de motifs séquentiels dans des bases de séquences. Ce type de données séquentielles est présent dans de nombreux domaines d'applications, comme par exemple l'analyse de séquences d'achats dans la grande distribution, l'analyse de séquences d'ADN en biologie ou encore l'étude des traces d'usagers sur des sites Web. Le domaine de la finance est un autre domaine d'application important possédant ce type de données. Celui-ci a représenté le contexte principal d'application avec le traitement de données financières où l'évolution de produits boursiers est représentée par des séquences d'événements. Ces données ont été fournies par la Caisse des Dépôts et Consignations, partenaire industriel de cette thèse CIFRE, et leur traitement visait à construire une représentation synthétique de différentes tendances boursières sous forme de motifs séquentiels caractéristiques [LB02b, LB02a, Lel01].

L'étude des différentes techniques d'extraction de motifs séquentiels (cf. chapitre 2) a donc permis d'identifier trois familles d'algorithmes distinctes : les approches dites *APriori* dont les principaux représentants sont les algorithmes GSP [SA96] et PSP [MCP98], celles dites par *Projection* (PrefixSpan [PHMAP01]) et enfin une dernière famille d'algorithmes, appelée *Listes d'occurrences*, représentée par l'algorithme *Spade* [Zak01]. Cette dernière famille se compose d'algorithmes qui comptent parmi les plus performants du domaine à faible support, et qui permettent d'intégrer certaines contraintes en plus de la contrainte de fréquence minimum. Les algorithmes de la première famille (*A Priori*), sont moins efficaces, d'après la littérature, sur des extractions effectuées à faibles supports et ceux de la deuxième famille (par *Projection*) ne permettent pas, à l'époque, d'intégrer des contraintes supplémentaires, telles des contraintes temporelles (cf. chapitre 2, section 2.2.2).

Dans le cadre de l'étude sur des données boursières, l'algorithme *Spade* [Zak01]

a donc été développé, ainsi que sa version contrainte, l'algorithme *cSpade* [Zak00], afin de caractériser les différentes classes de tendances. Les premières expérimentations consommaient énormément de mémoire, empêchant parfois les extractions d'aboutir. L'étude des structures de données manipulées par ces algorithmes (i.e., listes d'occurrences) lors de leur exécution, a confirmé que ces listes étaient trop longues pour pouvoir tenir en mémoire. Cependant, celle-ci a aussi permis de faire ressortir le fait que les occurrences contenues dans les listes représentaient souvent des apparitions consécutives des items dans les données.

La présence de telles répétitions pouvait être expliquée par les faibles variations qu'elles présentaient. Celles-ci n'étaient pas forcément liées à ce jeu de données, puisque le même phénomène a été identifié dans d'autres domaines d'application, où les séquences étaient en partie construites à partir de données quantitatives ou encore à partir d'alphabets de petite taille, comme les séquences d'ADN en biologie (cf. figure 1.4, page 21).

Il semblait alors possible et intéressant de pouvoir décrire de façon plus compacte ces occurrences consécutives, tout en étant capable de réaliser les calculs nécessaires aux extractions.

Une structure d'occurrences généralisées a donc été développée dans le cas de recherche de motifs séquentiels prenant en compte uniquement la contrainte de fréquence minimum. Ceci a abouti à l'algorithme *GoSpade* [LRBE03b], présenté au chapitre 3, section 3.2, dont il a été démontré que les résultats qu'il retourne sont corrects. De plus, des expérimentations effectuées sur des jeux de données réelles et synthétiques, ont montré que cet algorithme offre des temps d'exécution bien plus intéressants que *Spade* [Zak01] qui travaille sur des listes d'occurrences classiques, lorsque les données contiennent des répétitions consécutives.

Les possibilités d'application d'un tel compactage dans le cas d'extractions faisant intervenir des contraintes temporelles (contrainte de fenêtre de temps maximum, contraintes de gap minimum et de gap maximum) ont ensuite été étudiées. La situation était plus complexe et a nécessité la mise au point d'occurrences généralisées plus sophistiquées. En effet, il était nécessaire de conserver des informations supplémentaires afin de pouvoir tester la satisfaction des différentes contraintes par les motifs, tout en permettant une réduction significative de la taille des listes d'occurrences. Une nouvel algorithme, *GoSpec* [LRBE03a], a été mis au point permettant de gérer ces occurrences généralisées contraintes. Il a aussi été prouvé que cet algorithme est juste et complet et plusieurs expérimentations, visant à le comparer avec l'algorithme *cSpade* [Zak00], ont permis de montrer son intérêt en pratique en présence de répétitions consécutives dans les données.

Comme beaucoup d'algorithmes d'extractions de motifs séquentiels, les algorithmes présentés, *GoSpade* et *GoSpec* (cf. chapitre 3), ont de fortes consommations de ressources en termes d'occupation mémoire et de nombre d'opérations,

rendant toute implémentation naïve inefficace en pratique. Il était donc intéressant de travailler sur des techniques d'optimisation appropriées, décrites dans la première partie du chapitre 4.

Nous avons ensuite présenté l'application de cette approche (i.e., occurrences généralisées), sur des données financières réelles. Les données financières considérées concernaient l'évolution de trois valeurs relatives au cours de *France Telecom* durant une année. Celles-ci ont été pré-traitées afin de pouvoir être représentées sous forme de séquences d'événements. Le principe consistait à effectuer une discrétisation des différentes valeurs considérées, afin d'obtenir un alphabet qui traduisait d'une part le sens de variation de la valeur et d'autre part l'intensité de cette variation.

Plusieurs groupes de séquences ont été isolés, correspondant respectivement à trois tendances boursières : *hausse* du marché, *baisse* du marché et *stabilité* du marché. Pour chaque groupe, une collection de motifs séquentiels fréquents a pu être extraite et il a été montré que ces collections étaient porteuses d'informations spécifiques au groupe qu'elles représentaient.

Ce résultat nous permet de conclure que l'utilisation de motifs séquentiels pour caractériser des tendances offre un potentiel intéressant pour aider les experts du domaine en complétant leurs outils classiques d'aide à la décision. Toutefois, l'incorporation de la connaissance extraite, sous la forme de motifs séquentiels fréquents, dans un tel processus décisionnel, nécessite encore des efforts spécifiques (e.g., présentation des connaissances sous forme synthétique, prise en compte d'autres contraintes, d'autres indices boursiers ou encore introduction d'informations qualitatives telles des dépêches de presse financières, dans les séquences).

En ce qui concerne les techniques d'extraction de motifs fréquents eux-mêmes (algorithmes présentés au chapitre 3), plusieurs évolutions possibles ont été identifiées et mériteraient d'être étudiées :

Spécialisation de *GoSpec* :
A propos des contraintes de temps gérées par *GoSpec*, certaines améliorations pourraient être apportées. En effet, même si la prise en compte des trois contraintes de temps simultanément a pour effet de réduire l'espace de recherche et le temps d'exécution total, il est clair que le processus devient plus complexe. Celui-ci pourrait être simplifié, au cas par cas, lorsque certaines contraintes ou combinaisons de contraintes n'apparaissent pas dans une extraction. Par exemple, il n'est plus nécessaire de conserver l'information sur la date de début des motifs (champs $tBeg$) si les extractions ne font pas intervenir de contrainte de fenêtre de temps maximum. De même, le champs $gmax$ devient inutile dès lors que les extractions sont effectuées sans la définition d'une contrainte de gap maximum. Cette adaptation du

contenu des listes permettrait de réduire la taille des occurrences généralisées contraintes, et aussi celle des listes puisque, comme on a pu le voir, ces informations nécessitent de conserver plus d'occurrences généralisées (e.g., occurrences généralisées concernant plusieurs dates de début et occurrences ne respectant pas le gap maximum).

Ajout d'autres contraintes :

L'algorithme *GoSpec* (Chapitre 3, section 3.3) permet d'effectuer des extractions en faisant intervenir une contrainte de fréquence minimum et des contraintes de temps. Cet algorithme peut être facilement adapté à la prise en compte de contraintes syntaxiques comme la présence ou l'absence de certains items dans la séquence. Cependant, la gestion de contraintes exprimées sous forme d'expressions régulières et de contraintes d'agrégats (e.g.,*moyenne*, *somme*), qui représentent des contraintes intéressantes du point de vue de la signification des résultats retournés, n'est pas aussi directe. C'est pourquoi, il pourrait être intéressant de faire évoluer ces algorithmes afin qu'ils puissent intégrer efficacement ce type de contraintes.

D'autre part, les deux algorithmes proposés (*GoSpade* et *GoSpec*, cf. chapitre 3) sont particulièrement bien adaptés à la présence de répétitions consécutives dans les données, et la représentation par listes d'occurrences généralisées permet de faire apparaître clairement ces répétitions. On peut se demander si la connaissance de cette information ne pourrait pas, d'une façon ou d'une autre, être utilisée afin d'accélérer le processus d'extraction (e.g., lors de la génération de motifs dont le dernier événement se répète) ou encore pour permettre l'expression de nouvelles contraintes. Par exemple, des contraintes qui limiteraient la recherche aux seuls motifs fréquents présentant au moins n répétitions successives de chaque événement, auquel cas, les occurrences généralisées dont l'intervalle de fin ne comprend pas n occurrences successives seraient supprimées.

Recherche d'autres motifs :

Malgré les efforts fournis pour améliorer les temps de calculs, la plupart des algorithmes d'extraction de motifs séquentiels présentent une limite inhérente au nombre de motifs. Certains travaux [YHA03, HSDT01], inspirés d'approches proposées pour la recherche de motifs fréquents dans des données sans composante temporelle ([PBTL99, PHM00, ZH02]), ont eu pour objectif de réduire l'ensemble des motifs recherchés à un sous-ensemble de motifs, dits *clos*. En pratique, la taille de ces ensembles de motifs est beaucoup plus petite, ce qui permet de réduire les temps d'extraction. De plus, ils présentent la particularité de permettre la génération de tous les motifs fréquents et de leur fréquence en un temps raisonnable. Il serait intéressant de réfléchir à l'adaptation de nos algorithmes pour la recherche de tels motifs.

Autres contextes d'application :

Il n'est pas rare que les bases de données utilisées soient mises à jour régulièrement, certaines techniques d'extraction des motifs, dites *incrémentales* (e.g., [MPT00, PZOD99, LL98]) ont été étudiées afin de recalculer rapidement l'ensemble des motifs fréquents à mesure que le contenu des bases de séquences évoluent, sans avoir à effectuer de nouveau un processus complet. L'évolution de nos algorithmes vers ce type d'extractions pourrait représenter un autre point intéressant.

D'autre part, pour certains domaines d'application, il existe une connaissance exprimée sous la forme d'une hiérarchie des attributs. Par exemple, dans le cas de séquences d'achats, les attributs peuvent être regroupés en catégories de produits, celles-ci étant elles-mêmes regroupées en catégories plus générales. Cette connaissance peut être utilisée pour effectuer des extractions à plusieurs niveaux d'abstraction, comme décrit dans [SA96]). Intuitivement, le fait de remplacer les items d'une séquence par des items de niveaux plus généraux, augmente le nombre de répétitions consécutives. Les algorithmes développés dans ce mémoire étant particulièrement bien adaptés à cette situation, il semble intéressant d'étudier leur adaptation à ce type d'extraction.

Autres domaines d'application :

Comme on a pu le mentionner à plusieurs reprises, le traitement de données séquentielles concerne de nombreux domaines d'application. Parmi ceux-ci, le domaine de la biologie en est un très actif ayant déjà donné lieu à de nombreux travaux ([And03, Sag00]). La majorité des techniques développées sont très spécifiques au domaine et très peu d'entre elles exploitent des techniques d'extraction de motifs séquentiels à proprement dites. Il serait alors intéressant d'étudier l'intérêt que pourrait avoir ces approches sur ce domaine particulier, notamment en utilisant les possibilités de prise en compte efficace de la présence des répétitions consécutives.

Bibliographie

[AAP01] Agarwal R., Aggarwal C., Prasad V. *A tree projection algorithm for finding frequent itemsets.* **In** : J. Parallel Distrib. Comput., 2001, vol. 61, n° 3, pp. 350–371.

[AFGY02] Ayres J., Flannick J., Gehrke J., et al. *Sequential pattern mining using bitmap representation.* **In** : *Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Edmonton, Alberta, Canada, July 2002, pp. 429–435.

[AHKV97] Ahonen H., Heinonen O., Klemettinen M., et al. *Applying data mining techniques in text analysis.* Tech. Rep. C-1997-23, University of Helsinki, Department of Computer Science, March 1997. 12 pages.

[ALB03] Albert-Lorincz A., Boulicaut J. *Mining frequent sequential patterns under regular expressions : a highly adaptative strategy for pushing constraints.* **In** : *Proc. (SIAMDM'03).* San Francisco, CA, USA, May 2003, pp. 316–320.

[AMS+96] Agrawal R., Mannila H., Srikant R., et al. *Fast discovery of association rules.* **In** : Fayyad U., al., Eds., *Advances in Knowledge Discovery in Data Mining.* IEEE Computer Society Press, 1996, pp. 307–328.

[And03] Andrade M. *Bioinformatics and Genomes.* Heidelberg, Germany :Springer-Verlag, janvier 2003, 250 pp.

[AS94a] Agrawal R., Srikant R. *Fast algorithms for mining association rules in large databases.* **In** : *Proc. of the 20th International Conference on Very Large Data Bases (VLDB'94).* Santiago de Chile, Chile, 1994, pp. 487–499.

[AS94b] Agrawal R., Srikant R. *Mining sequential patterns.* Tech. Rep. RJ 9910, IBM Almaden Research Center, San Jose, CA, USA, October 1994. 29 pages.

[AS95] Agrawal R., Srikant R. *Mining sequential patterns.* **In** : Yu P.S., Chen A.S.P., Eds., *Proc. of the 11th International Conference on Data Engineering (ICDE'95).* Taipei, Taiwan, 1995, pp. 3–14.

[BC96] BERNDT D.J., CLIFFORD J. *Finding patterns in time series : A dynamic programming approach.* **In** : FAYYAD U., PIATETSKY-SHAPIRO G., SMYTH P., et al., Eds., *Advances in Knowledge Discovery and Data Mining*. New York :AAAI/MIT Press, 1996, pp. 229–248.

[BGMP03a] BONCHI F., GIANNOTTI A., MAZZANTI A., et al. *Adaptative constraint pushing in frequent pattern mining.* **In** : *Proc. of the 7th European Conference on Principles and Pratice of Knowledge Discovery in Databases (PKDD'03).* LNCS 2838, Cavtat-Dubrovnik, Croatia, September 2003, pp. 47–58.

[BGMP03b] BONCHI F., GIANNOTTI A., MAZZANTI A., et al. *Exante : Anticipate data reduction in constrainted pattern mining.* **In** : *Proc. of the 7th European Conference on Principles and Pratice of Knowledge Discovery in Databases (PKDD'03).* LNCS 2838, Cavtat-Dubrovnik, Croatia, September 2003, pp. 59–70.

[BSWJL98] BETTINI C., SEAN WANG X., JAJODIA S., et al. *Discovering frequent event patterns with multiple granularities in time sequences.* **In** : Knowledge and Data Engineering, 1998, vol. 10, n° 2, pp. 222–237.

[BT01] BUHLER J., TOMPA M. *Finding motifs using random projections.* **In** : *Proc. of the 5th Annual Int. Conf. on Computational Biology (RECOMB'01).* ACM Press, Montreal, Quebec, Canada, April 2001, pp. 69–76.

[CMB02] CAPELLE M., MASSON C., BOULICAUT J. *Mining frequent sequential patterns under a similarity constraint.* **In** : LNCS, Ed., *Proc. of the 3rd Int. Conference on Intelligent Data Engineering and Automated Learning (IDEAL'02).* Manchester, UK, August 2002, vol. 2412, pp. 1–6.

[CMS97] COOLEY R., MOBASHER B., SRIVASTAVA J. *Web mining : Information and pattern discovery on the world wide web.* **In** : *Proc. of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97).* Newport Beach, CA, 1997, pp. 558–567.

[Coo00] COOLEY R. *Web Usage Mining : Discovery and Application of Interesting Patterns from Web Data.* Ph.D. thesis, University of Minnesota, May 2000. 180 pages.

[DK01] DE RAEDT L., KRAMER S. *The levelwise version space algorithm and its application to molecular fragment finding.* **In** : *Proceedings IJCAI'01.* Morgan Kaufmann, Seattle, USA, août 2001, pp. 853 – 862.

[DL99] DONG G., LI J. *Efficient mining of emerging patterns : Discovering trends and differences.* **In** : *Proc. of the 5th ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining (KDD'99). ACM Press, San Diego, CA, USA, August 1999, pp. 43–52.

[FPSM91] FRAWLEY W., PIATETSKY-SHAPIRO G., MATHEUS C. *Knowledge discovery in databases : an overview*. **In** : PIATETSKY-SHAPIRO G., FRAWLEY W., Eds., *Knowledge in Discovery in Databases*. Menlo Park, CA, USA, 1991, pp. 1–27.

[FPSS96a] FAYYAD U., PIATETSKY-SHAPIRO G., SMYTH P. *The KDD process for extracting useful knowledge from volumes of data.*. **In** : Communications of the ACM, nov. 1996, vol. 39, n° 11, pp. 27 – 34.

[FPSS96b] FAYYAD U., PIATETSKY-SHAPIRO G., SMYTH P. *Knowledge discovery and data mining : Towards a unifying framework*. **In** : *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*. Portland, Oregon, August 1996, pp. 82–88.

[FPSSU96] FAYYAD U., PIATETSKY-SHAPIRO G., SMYTH P., et al. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[Gar99] GARDARIN G. *Internet/Intranet et Bases de Données : Data Web, Data Media, Data Warehouse, Data Mining*. Paris :Eyrolles, 1999, 264 pp.

[GGK01] GURALNIK V., GARG N., KARYPIS G. *Parallel tree projection algorithm for sequences mining*. **In** : *Proc. of the European Conference on Parallel Processing*. 2001, pp. 310–320.

[GNPS03] GIANNOTTI F., NANNI M., PEDRESCHI D., et al. *Webcat : Automatic categorization of web search results*. **In** : *(Proceedings of the 11th Italian Symposium on Advanced Database Systems SEBD'03)*. Rubettino Editore, Cetraro, Italy, June 2003, pp. 507–518.

[GRK99] GAROFALAKIS M., RASTOGI R., K. S. *Spirit : Sequential pattern mining with regular expression constraints*. **In** : *Proc. of the 25th International Conference on Very Large Databases (VLDB'99)*. Edinburgh, United Kingdom, September 1999, pp. 223–234.

[GRSS99] GAROFALAKIS M., RASTOGI R., SESHADRI S., et al. *Data mining and the web : Past, present and future*. **In** : *Workshop on Web Information and Data Management*. 1999, pp. 43–47.

[Gus97] GUSFIELD D. *Algorithms on String, Trees, and Sequences*. New York :Cambridge University Press, 1997, 534 pp.

[HDY99] HAN J., DONG G., YIN Y. *Efficient mining of partial periodic patterns in time series database*. **In** : *Proc. of the 15th Int. Conf. on Data Engineering*. IEEE computer Society, Sydney, Australia, 1999, pp. 106–115.

[HHMAC+00] Han J. J. Pei, Han Mortazavi-Asl B., Chen Q., et al. *Freespan : Frequent pattern-projected sequential pattern mining.* **In** : *Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00).* Boston, MA, USA, August 2000, pp. 355–359.

[HK00] Han J., Kamber M. *Data Mining : Concepts and Techniques.* William College, USA : Morgan Kaufmann, 2000, 550 pp.

[HKM+96] Hätönen K., Klemettinen M., Mannila H., et al. *Tasa : Telecomunications alarm sequence analyzer or : How to enjoy faults in your network.* **In** : *1996 IEEE Network Operations and Management Symposium (NOMS'96).* Kyoto, Japan, April 1996, pp. 520–529.

[HPY00] Han J., Pei J., Yin Y. *Mining frequent patterns without candidate generation.* **In** : *Proceedings ACM SIGMOD 2000.* Dallas, USA, mai 2000, pp. 1–12.

[HSDT01] Harms S., Saquer J., Deogun J., et al. *Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints..* **In** : *Proc. of the IEEE Int. Conf. on Data Mining (ICDM'01).* Silicon Valley, CA, November 2001, pp. 603–606.

[IM96] Imielinski T., Mannila H. *A database perspective on knowledge discovery.* **In** : Communications of the ACM, nov. 1996, vol. 39, n° 11, pp. 58–64.

[Jeu02] Jeudy B. *Optimisation de requêtes inductives : application à l'extraction sous contraintes de règles d'association.* Ph.D. thesis, Institut National des Sciences Appliquées de Lyon, décembre 2002. 138 pages.

[KB00] Kosala R., Blockeel H. *Web mining research : A survey.* **In** : SIGKDD : SIGKDD Explorations : Newsletter of the Special Interest Group (SIG) on Knowledge Discovery and Data Mining, ACM, 2000, vol. 2, pp. 1–15.

[KD01] Kramer S., De Raedt L. *Feature construction with version spaces for biochemical applications.* **In** : *Proceedings ICML'01.* Morgan Kaufmann, William College, USA, juil. 2001, pp. 258 – 265.

[Klm99] Klmemettinen M. *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases.* Ph.D. thesis, Department of Computer Science - University of Helsinki, january 1999. 138 pages.

[KPWD03] Kum H., Pei J., Wang W., et al. *ApproxMAP : Approximate mining of consensus sequential patterns.* **In** : *Proc. of the 3rd SIAM Int. Conf. on Data Mining (SDM).* 2003, pp. 311–315.

[LAS97] LENT B., AGRAWAL R., SRIKANT R. *Discovering trends in text databases.* **In** : *Proc. of the 3rd International Conference on Knowledge Discovery in Databases and Data Mining (KDD'97).* AAAI Press, Newport beach, CA, USA, August 1997, pp. 227–230.

[LB02a] LELEU M., BOULICAUT J.F. *Signature de situations boursières représentées par des séquences d'événements.* **In** : *Actes des Journées Francophones d'Extraction et de Gestion des Connaissances (EGC'02).* Montpellier, France, Januray 2002, pp. 89–100.

[LB02b] LELEU M., BOULICAUT J.F. *Signing stock market situations by means of characteristic sequential patterns.* **In** : *Proc. of the 3rd International Conference on Data Mining (DM'02).* Bologna, Italy, September 2002, pp. 655–664.

[LDR02] LEE S., DE RAEDT L. *Constraint based mining of first-order sequences in seqlog.* **In** : DŽEROSKI S., RAEDT L.D., WROBEL S., Eds., *Proc. of the 1st SIGKDD Workshop on Multi-Relational Data Mining (MRDM'02).* University of Alberta, Edmonton, Canada, July 2002, pp. 80–96.

[Lel01] LELEU M. *Recherche de cas similaires en présence de données qualitatives et quantitatives par l'extraction de motifs séquentiels.* Poster à la conférence d'Apprentissage Cap'01 organisée par l'Association Française pour l'Intelligence Artificielle, June 2001. 2 pages, Grenoble, France.

[LL98] LIN M., LEE S. *Incremental update on sequential patterns in large databases.* **In** : *Proc. of the International Conference on Tools for Artificial Intelligence Conference (ICTAI'98).* May 1998, pp. 24–31.

[LRBE03a] LELEU M., RIGOTTI C., BOULICAUT J.F., et al. *Constrained-based mining of sequential patterns over datasets with consecutive repetitions.* **In** : *Proc.2003 Int. Conf. on Principles of Data Mining and Knowledge Discovery in Databases (PKDD'03).* LNCS 2838, Cavtat-Dubrovnik, Croatia, September 2003, pp. 303–314.

[LRBE03b] LELEU M., RIGOTTI C., BOULICAUT J.F., et al. *Go-spade : Mining sequential patterns over datasets with consecutive repetitions.* **In** : *Proc.2003 Int. Conf. Machine Learning and Data Mining (MLDM'03).* LNCS 2734, Leipsig, Germany, July 2003, pp. 293–306.

[MA01] MORTAZAVI-ASL B. *Discovering and Mining User Web-Page Traversal Patterns.* Master's thesis, Simon Fraser University, April 2001. 93 pages.

[Mas02] MASSEGLIA F. *Algorithmes et applications pour l'extraction de motifs séquentiels dans le domaine de la fouille de données : de*

l'incrémental au temps réel. Ph.D. thesis, Université de Versailles St Quentin, France, 2002. 248 pages.

[MCP98] MASSEGLIA F., CATHALAT F., PONCELET P. *The PSP approach for mining sequential patterns*. **In** : *Proc. of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery in Databases (PKDD'98)*. LNAI, Nantes, France, September 1998, vol. 1510, pp. 176–184.

[MJ02] MASSON C., JACQUENET F. *Mining frequent sequences with spirit-log*. **In** : IN ARTIFICIAL INTELLIGENCE L.N., Ed., *Proc. of the 1éth Int. Conference on Inductive Logic Programming (ILP'02)*. Syndney, Australia, July 2002, vol. 2583, pp. 166–181.

[MPC99] MASSEGLIA F., PONCELET P., CICCHETTI R. *An efficient algorithm for web usage mining*. **In** : Networking and Information Systems Journal, 1999, vol. 2, n° 5-6.

[MPT99] MASSEGLIA F., PONCELET P., TEISSEIRE M. *Extraction efficace de motifs séquentiels : le prétraitement des données*. **In** : *Actes des Journées Bases de Données Avancées (BDA'99)*. Bordeaux, France, Octobre 1999, pp. 341–360.

[MPT00] MASSEGLIA F., PONCELET P., TEISSEIRE M. *Incremental mining of sequential patterns in large databases*. **In** : *Proc. of 16èmes Journées Bases de Données Avancées (BDA'00)*. Blois, France, October 2000, pp. 345–366.

[MT96] MANNILA H., TOIVONEN H. *Discovery generalized episodes using minimal occurrences*. **In** : *Proc. of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*. Portland, Oregon, August 1996, pp. 146–151.

[MT97] MANNILA H., TOIVONEN H. *Levelwise search and borders of theories in knowledge discovery*. **In** : Data Mining and Knowledge Discovery, 1997, vol. 1, n° 3, pp. 241–258.

[MTV95] MANNILA H., TOIVONEN H., VERKAMO I. *Discovering frequent episodes in sequences*. **In** : *Proc. of the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95)*. AAAI Press, Montreal, Canada, August 1995, pp. 210–215.

[MTV97] MANNILA H., TOIVONEN H., VERKAMO A. *Discovery of frequent episodes in event sequences*. **In** : Data Mining and Knowledge Discovery, November 1997, vol. 1, n° 3, pp. 259–298.

[PBTL99] PASQUIER N., BASTIDE Y., TAOUIL R., et al. *Efficient mining of association rules using closed itemset lattices*. **In** : Information Systems, 1999, vol. 24, n° 1, pp. 25–46.

[PCGS02] PISANTI N., CROCHEMORE M., GROSSI R., et al. *A basis for repeated motifs in pattern discovery and text mining*. Internal report, University of Marne-la-Valée, July 2002. 11 pages.

[PHM00] PEI J., HAN R., MAO R. *Closet : An efficient algorithm for mining frequent closed itemsets*. **In** : D. GUNOPULOS R.R., Ed., *Proc. of the 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'00)*. Dallas, TX, may 2000, pp. 21–30.

[PHMAP01] PEI J., HAN B., MORTAZAVI-ASL B., et al. *Prefixspan : Mining sequential patterns efficiently by prefix-projected pattern growth*. **In** : *Proc. of the 17th International Conference on Data Engineering (ICDE'01)*. 2001, pp. 215–224.

[PHW02] PEI J., HAN B., WANG W. *Mining sequential patterns with constraints in large databases*. **In** : *Proc. of the 11th International Conference on Information and Knowledge Management (CIKM'02)*. McLean, VA, USA, November 2002, pp. 18–25.

[Pov99] POVINELLI R. *Time Series Data Mining : Identifying Temporal Patterns for Characterization and Prediction of Time Series Events*. Ph.D. thesis, Marquette University, 1999. 183 pages.

[PS00] PEVZNER P., SZE S.H. *Combinatorial approaches to finding subtle signals in dna sequences*. **In** : *Proc. of the 8th Int. Conf. on Intelligent Systems for Molecular Biology*. San Diego, CA, USA, August 2000, pp. 269–278.

[PZOD99] PARTHASARATHY S., ZAKI M., OGIHARA M., et al. *Incremental and interactive sequence mining*. **In** : *Proc. of the 8th International Conference on Information and Knowledge Management (CIKM'99)*. Kansas City, MO, USA, November 1999, pp. 251–258.

[SA96] SRIKANT R., AGRAWAL R. *Mining sequential patterns : Generalizations and performance improvements*. **In** : *Proc. of the 5th International Conference on Extending Database Technology (EDBT'96)*. Avignon, France, September 1996, pp. 3–17.

[Sag00] SAGOT M. *Combinatorial algorithms in computational biology*. **In** : Ercim News, october 2000, vol. 1, n° 43, pp. 27–38.

[SK02] SENO M., KARYPIS G. *SLPminer : An algorithm for finding frequent sequential patterns using length-decreasing support constraint*. **In** : *Proc. of the 2002 IEEE International Conference on Data Mining (ICDM'02)*. Maebashi City, Japan, December 2002, pp. 418–425.

[YHA03] YAN X., HAN J., AFSHAR R. *CloSpan : Mining closed sequential patterns in large databases*. **In** : *Proc. 2003 SIAM International*

Conference on Data Mining (SDM'03). San Francisco, CA, May 2003, pp. 53–65.

[Zak98] ZAKI M. *Efficient enumeration of frequent sequences*. **In** : *Proc. of the 7th International Conference on Information and Knowledge Management (CIKM'98)*. November 1998, pp. 68–75.

[Zak00] ZAKI M. *Sequence mining in categorical domains : incorporating constraints*. **In** : *Proc. of the 9th International Conference on Information and Knowledge Management (CIKM'00)*. Washington, DC, USA, November 2000, pp. 422–429.

[Zak01] ZAKI M. *Spade : an efficient algorithm for mining frequent sequences*. **In** : Machine Learning, Special issue on Unsupervised Learning, Jan/Feb 2001, vol. 42, n° 1/2, pp. 31–60.

[ZH02] ZAKI M., HSIAO C. *CHARM : An efficient algorithm for closed itemset mining*. **In** : *Proc. of the 2nd SIAM International Conference on Data Mining*. Arlington, Virginia , USA, April 2002, pp. 117–128.

[ZLO98] ZAKI M., LESH N., OGIHARA M. *PlanMine : Sequence mining for plan failures*. **In** : AGRAWAL R., STOLORZ P., PIATETSKY-SHAPIRO G., Eds., *Proc. of the 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98)*. New York, NY, 1998, pp. 369–373.