UNIVERSITÉ D'ARTOIS

Université Ferhat Abbas Sétif 1

École Doctorale Sciences Technologies et Santé n°585
Unité de Recherche CRIL

Thèse de doctorat

Présentée en vue de l'obtention du grade de
Docteur en Informatique

de l'Université d'Artois
et
de l'Université de Ferhat Abbas-Sétif 1

par

Ikram Nekkache

Extraction des motifs fréquents sous contraintes

Soutenue le 5 décembre 2021 après avis des rapporteurs, devant le jury d'examen :

M^me^ Z. Aliouat, Professeur, Université de Ferhat Abbas-Sétif 1	**Président**
M. B. Benhamou, Maître de Conférences HDR, Aix-Marseille Université	**Rapporteur**
M^me^ H. Drias, Professeur, Université des Sciences et de la Technologie Houari Boumediene	**Rapporteure**
M. L. Sais, Professeur, Université d'Artois	**Directeur de thèse**
M^me^ N. Kamel, Professeur, Université de Ferhat Abbas-Sétif 1	**Co-directrice**
M. S. Jabbour, Maître de Conférences HDR, Université d'Artois	**Examinateur**

Mise en page avec memcril (B. Mazure, CRIL) et thloria (D. Roegel, LORIA).

Remerciements

Je tiens à remercier en premier lieu mes directeurs, Mme Nadjet Kamel et Mr. Lakhdar Sais, pour leur patience, leur confiance en moi et les pistes de recherches intéressantes proposées durant cette thèse.

Ma profonde reconnaissance revient à mon encadrant de thèse Mr. Said Jabbour, qui a suivi ce travail de près et de loin avec un grand intérêt. Merci pour votre encadrement et vos encouragements, qui m'ont permis de s'épanouir et de faire des progrès dans divers domaines. Merci pour votre soutien scientifique et moral durant les moments difficiles que j'ai rencontrés et surtout lors de la rédaction.

Je tiens à remercier les membres de jury d'avoir accepté de juger ce travail.

Je tiens également à remercier tous mes amis et collègues du CRIL pour les nombreuses intéressantes discussions que nous avons eu pendant les pauses-café.

Je tiens à exprimer ma plus profonde gratitude à mes chers parents, mes adorables frères et sœurs : Ramla, Kamilia, Youcef, Adem et Meriem qui m'ont encouragée et soutenue toute au long de mes études.

Je dédie cette thèse
à ma famille.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Ces dernières années, les données sont considérées comme une ressource précieuse. Grâce aux moyens fournis par la fouille de données, des connaissances peuvent être extraites des bases de données. Plus précisément, il s'agit d'un processus d'analyse des tuples d'une base de données pour établir des relations entre les données ou obtenir des motifs. Afin de prendre de meilleures décisions dans divers domaines, ces connaissances découvertes à partir des données ont été utilisées dans des applications importantes telles que la finance, les réseaux sociaux, la bioinformatique et le commerce. Différentes techniques ont été développées pour le problème de la fouille de motifs, à savoir les itemsets et les motifs séquentiels.

La fouille des motifs fréquents est une tâche fondamentale dans le domaine de la fouille de données. Elle consiste à extraire les motifs intéressants dans les bases de données en fonction des préférences de l'utilisateur. Ces motifs peuvent être constitués d'un ensemble simple d'attributs (appelés les itemsets), ou des motifs plus complexes (tels que les règles d'associations, les séquences et les graphes).

Souvent la fréquence est le critère dominant dans la fouille des itemsets. L'extraction des itemsets fréquents révèle des associations implicites et pertinentes entre les items dans les bases de données. Une première application concerne des transactions des produits achetés par des clients dans un magasin [2]. Cette application a permis d'extraire des associations entre les produits achetés. Ces associations servent à mieux comprendre le comportement des consommateurs afin d'établir de meilleures stratégies de marketing ou de meilleures recommandations pour les clients. En raison de l'importance de cette tâche, divers domaines d'application ont exploité la fouille des itemsets fréquents comme en bioinformatique [96], dans la classification des images [37], dans la détection de logiciels malveillants [33]..., etc. Par conséquent, plusieurs algorithmes ont été conçus, par exemple, nous citons l'algorithme Apriori [3], Eclat [145] et FP-growth [60]. Le développement de l'extraction des itemsets a conduit à l'extraction de règles d'associations, étendue aussi à l'extraction des motifs séquentiels.

La fouille des séquences implique la découverte des motifs séquentiels qui représentent des sous-séquences fréquentes à partir des bases de données de séquences. L'extraction des motifs séquentiels fréquents révèle des relations entre des items ordonnés. L'extraction de motifs séquentiels a été utilisée dans diverses applications. Par exemple, le domaine de la bioinformatique utilise des méthodes de la fouille de séquences pour l'identification et la prédiction de sous-séquences dans les séquences d'ADN [132] et de protéines [133], dans le domaine de la médecine pour identifier des motifs séquentiels intéressants à partir des parcours de soins des patients [113] et dans l'analyse des transactions commerciales [119]. Par conséquent, plusieurs algorithmes ont été proposés, comme l'algorithme GSP [120], SPADE [144] et PrefixSpan [60].

Cette thèse est divisée en deux parties principales. La première est réservée à la présentation de l'état de l'art, incluant divers concepts et algorithmes liés à nos contributions, impliquant principalement la satisfiabilité propositionnelle, la fouille des itemsets et la fouille des séquences. Dans la deuxième partie, nous avons introduit nos contributions à la fouille des itemsets en utilisant la satisfiabilité propositionnelle et dans la fouille des séquences en exploitant les symétries.

Cette thèse est structurée en cinq chapitres. Le premier chapitre est consacré à la logique propo-

sitionnelle et au problème de satisfiabilité propositionnelle utile pour la compréhension du deuxième chapitre.

Le deuxième chapitre porte sur la fouille des itemsets. Nous étudions d'abord certaines propriétés des itemsets. Puis, nous présentons les principales approches spécialisées et déclaratives proposées pour l'extraction des itemsets fréquents. Ensuite, nous montrons les représentations condensées les plus connues des itemsets. Enfin, nous donnons un aperçu de quelques extensions du problème d'extraction des itemsets.

Dans le troisième chapitre, nous présentons le problème de la fouille des séquences. Nous commençons par introduire les différentes notions liées aux séquences. Ensuite, nous présentons les principaux travaux pour extraire des motifs séquentiels fréquentes. Enfin, nous décrivons quelques variantes du problème d'extraction des séquences.

Dans le chapitre quatre, nous présentons notre première contribution qui est un encodage compacte basé sur la satisfiabilité propositionnelle pour l'extraction des itemsets. Ensuite nous montrons comment le problème de compression de l'encodage de la fouille des itemsets peut être vue comme un problème de compression de matrice booléenne. Enfin, des expérimentations sont effectuées pour montrer la réduction de la taille de l'encodage et son effet sur les performances.

Dans le dernier chapitre, avant de présenter notre deuxième contribution, nous donnons les différentes notions de symétries. Puis, nous proposons une méthode de détection des symétries dans les bases de séquences. Ensuite, nous présentons deux approches dédiées à l'exploitation de ces symétries pour l'extraction de motifs séquentiels. La première approche consiste à intégrer les symétries détectées dans un algorithme de type Apriori. Dans la deuxième approche, nous proposons d'exploiter les symétries durant une étape de prétraitement en éliminant les symétries existantes dans les bases de séquences. Enfin, à travers des expérimentations, nous montrons que l'exploitation des symétries améliore le processus d'extraction de motifs séquentiels.

Nous terminons ce manuscrit par une conclusion générale.

Ce travail a donné lieu aux publications suivantes :

— Detecting and exploiting symmetries in sequential pattern mining in IJDMMM « International Journal of Data Mining, Modelling and Management » 2021.

— Towards a Compact SAT-Based Encoding of Itemset Mining Tasks in CPAIOR « International Conference on Artificial Intelligence and Operation Research» 2021.

Première partie

État de l’art

Chapitre 1

Logique propositionnelle & SAT

Sommaire

Ce chapitre introduit le concept de la logique propositionnelle, ensuite il introduit la définition du problème de satisfiabilité (SAT) et ses algorithmes de résolution, à savoir les procédures DPLL et CDCL. À la fin de ce chapitre, l'énumération de toutes les solutions du problème SAT est présentée.

1.1 Logique propositionnelle

La logique propositionnelle comporte deux aspects : le premier aspect concerne la syntaxe et le deuxième aspect est la sémantique. Afin de comprendre la logique propositionnelle, nous expliquons d'abord sa syntaxe.

1.1.1 Syntaxe

L'aspect syntaxique de la logique propositionnelle est formellement défini comme suit :

Définition 1 (Atome)

Une proposition atomique est une variable booléenne, qui prend une valeur $faux$ ou $vrai$. La valeur $faux$ est représentée par le nombre 0 et la valeur $vrai$ est représentée par le nombre 1. Une proposition atomique est indivisible. Nous utilisons les lettres minuscules $p, q, r, etc.$ pour désigner une proposition.

Définition 2 (Langage propositionnel)

Soit P un ensemble infini dénombrable de variables propositionnelles. Le langage de la logique propositionnelle est formé à partir d'un sous-ensemble de variables propositionnelles de P, des parenthèses $\{(,)\}$, des connecteurs logiques $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ et des constantes booléennes $\{\top, \bot\}$.

Définition 3 (Formule propositionnelle)

Une formule propositionnelle est une proposition sous la forme de l'une des expressions suivantes. Nous utilisons les lettres grecques Σ, Φ, Δ, etc. pour désigner une formule propositionnelle.

- Un atome est une formule ;
- Les constantes booléennes $\top$ et $\bot$ sont des formules ;
- La négation d'une formule Σ notée $\neg\Sigma$;
- La conjonction de deux formules Σ et Φ notée $(\Sigma \wedge \Phi)$;
- La disjonction de deux formules Σ et Φ notée $(\Sigma \vee \Phi)$;
- L'implication de deux formules Σ et Φ notée $(\Sigma \rightarrow \Phi)$;
- L'équivalence entre deux formules Σ et Φ notée $(\Sigma \leftrightarrow \Phi)$.

Exemple

Soit $\{p, q, r, s\}$ un ensemble de variables propositionnelles, alors $((p \vee q) \rightarrow (r \vee (\neg s)))$ est une formule bien formée, contrairement à $((p \vee q) \rightarrow (\vee(\neg s)))$.

Définition 4 (littéral)

Un littéral est une variable propositionnelle p, appelé littéral positif, ou sa négation $\neg p$, appelé littéral négatif. Le littéral p (resp. $\neg p$) est dit complémentaire du littéral $\neg p$ (resp. p).

Soit Σ une formule propositionnelle, l'ensemble des variables propositionnelles de Σ est noté par $\mathcal{V}_\Sigma$.

Après avoir défini la syntaxe de la logique propositionnelle, nous nous intéressons à sa sémantique dans la section suivante.

1.1.2 Sémantique

En ce qui concerne l'aspect sémantique de la logique propositionnelle, l'interprétation d'une formule propositionnelle est essentielle pour déterminer la vérité d'une formule. Elle est définie comme suit :

Définition 5 (Interprétation des variables)

Une interprétation des variables μ est une application qui affecte à toute variable une valeur de vérité $\{0, 1\}$.

L'interprétation des variables propositionnelles peut être étendue aux formules propositionnelles en utilisant les règles suivantes :

— $\mu(\bot) = 0$;
— $\mu(\top) = 1$;
— $\mu(\neg\Sigma) = 1$ si et seulement si $\mu(\Sigma) = 0$;
— $\mu(\Sigma \vee \Phi) = 1$ si et seulement si $\mu(\Sigma) = 1$ ou $\mu(\Phi) = 1$;
— $\mu(\Sigma \wedge \Phi) = 1$ si et seulement si $\mu(\Sigma) = 1$ et $\mu(\Phi) = 1$;
— $\mu(\Sigma \rightarrow \Phi) = 0$ si et seulement si $\mu(\Sigma) = 1$ et $\mu(\Phi) = 0$;
— $\mu(\Sigma \leftrightarrow \Phi) = 1$ si et seulement si $\mu(\Sigma) = \mu(\Phi)$.

L'interprétation μ d'une formule est généralement exprimée sous la forme d'un ensemble de littéraux tels que $p \in \mu$ si $\mu(p) = 1$ et $\neg p \in \mu$ si $\mu(p) = 0$. Une interprétation peut être partielle ou complète comme définit ci-dessous :

Définition 6 (Interprétation partielle et complète)

Soit Σ une formule propositionnelle. Une interprétation μ définie sur l'ensemble de variables de Σ est appelée :
— Partielle si $|\mu| < |\mathcal{V}_\Sigma|$;
— Complète si $|\mu| = |\mathcal{V}_\Sigma|$.

Dans la suite de ce manuscrit, si aucune information sur la nature de l'interprétation n'est fournie, celle-ci est considérée comme complète.

À partir d'une interprétation μ, la sémantique des connecteurs logiques, comme indiqué dans le tableau 1.1, est utilisée pour déterminer la valeur de vérité de toute formule.

Σ	Φ	$\neg\Sigma$	$(\Sigma \vee \Phi)$	$(\Sigma \wedge \Phi)$	$(\Sigma \Rightarrow \Phi)$	$(\Sigma \Leftrightarrow \Phi)$	$(\Sigma \oplus \Phi)$
1	1	0	1	1	1	1	0
1	0	0	1	0	0	0	1
0	1	1	1	0	1	0	1
0	0	1	0	0	1	1	0

TABLE 1.1 – Sémantique des connecteurs logiques usuels.

La valeur de vérité d'une formule est obtenue en étendant celle des variables comme décrit précédemment. Par conséquent, la sémantique d'une formule dans une interprétation μ est définie par la valeur de vérité des variables qui composent μ.

Exemple

Soient $\Sigma = (p \vee q) \wedge (p \Leftrightarrow r)$ une formule propositionnelle et $\mu = \{p, \neg q, r\}$ une interprétation. Nous avons $\mu(\Sigma) = \mu(1 \vee 0) \wedge \mu(1 \Leftrightarrow 1) = \mu(1 \wedge 1) = 1$.

Définition 7 (Modèle)

Une interprétation μ est dite modèle de Σ, noté $\mu \models \Sigma$, si et seulement μ rend Σ vraie, c'est-à-dire : $\mu(\Sigma) = 1$.

Dans la suite, nous notons l'ensemble des modèles d'une formule Σ par $\mathcal{M}(\Sigma)$.

Définition 8 (Formules équivalentes)

Deux formules propositionnelles Σ et Φ sont dites équivalentes, noté $\Sigma \equiv \Phi$, si et seulement si $\mathcal{M}(\Sigma) = \mathcal{M}(\Phi)$.

Définition 9 (Formule satisfiable)

Une formule Σ est dite satisfiable (ou cohérente) (ou consistante) si elle admet au moins un modèle.

Définition 10 (Formule insatisfiable)

Une formule Σ est dite insatisfiable (ou incohérente) (ou contradictoire) (ou inconsistante) si elle n'admet aucun modèle.

Définition 11 (Formule valide)

Une formule est valide (ou tautologie) si toute interprétation de Σ est un modèle de Σ.

La suite de cette section est consacrée à la définition des différentes formes normales définies pour représenter les formules propositionnelles.

1.1.3 Formes normales

La forme normale négative (NNF), la forme normale conjonctive (CNF) et la forme normale disjonctive (DNF) sont les trois formes particulières les plus utilisées.

Définition 12 (Monôme)

Un monôme υ est une conjonction finie de littéraux distincts. En d'autres termes, un monôme est de la forme $\upsilon = (p_1 \wedge p_2 \wedge \cdots \wedge p_n)$ où chaque p_i est un littéral.

Définition 13 (Clause)

Une clause ω est une disjonction finie de littéraux distincts. C'est-à-dire une clause est de la forme $\omega = (p_1 \vee p_2 \vee \cdots \vee p_n)$ où chaque p_i est un littéral.

Définition 14 (Variantes de clauses)

- **clause unitaire** est une clause constituée seulement d'un seul littéral ;
- **clause binaire** est une clause qui contient exactement deux littéraux ;
- **clause de Krom** est une clause qui contient au plus deux littéraux ;
- **clause positive (resp. négative)** est une clause qui ne contient pas de littéraux négatifs (resp. positifs) ;
- **clause mixte** est une clause constituée de littéraux positifs et négatifs ;
- **clause vide** est une clause notée $\bot$ et ne contenant aucun littéral. Elle est par définition insatisfiable ;
- **clause tautologique** est une clause contenant un littéral et son complémentaire. Elle est par définition vraie ;
- **clause de Horn (resp. clause reverse Horn)** est une clause qui contient au plus un littéral positif (resp. négatif).

Définition 15 (Forme NNF)

Une formule Σ est sous forme normale négative (NNF), noté NNF(Σ), si elle se compose uniquement de conjonctions, de disjonctions et de littéraux.

Exemple

Soit Σ la formule définie sur les littéraux $\{p, q, r\}$ comme suit : $\Sigma = r \wedge (\neg p \vee q)$. Σ est une NNF(Σ).

Définition 16 (Forme CNF)

Une formule Σ est sous forme normale conjonctive (CNF), notée CNF(Σ), si et seulement si Σ est une conjonction de clauses. Une formule Σ en CNF est définie formellement comme suit :

$$CNF(\Sigma) = \bigwedge_{\omega \in \Sigma} \left(\bigvee_{p_i \in \omega} p_i \right)$$

Exemple

Soit $\{p, q, r\}$ un ensemble de littéraux et la formule $\Sigma = (p \vee q) \wedge (q \vee \neg r)$. Σ est la conjonction des deux clauses $(p \vee q)$ et $(q \vee \neg r)$. Par conséquent Σ est une CNF(Σ).

Définition 17 (Forme DNF)

Une formule Σ est sous forme normale disjonctive (DNF), noté DNF(Σ) si et seulement si Σ est une disjonction de monômes. Une formule Σ en DNF peut être représentée comme suit :

$$DNF(\Sigma) = \bigvee_{v \in \Sigma} \left(\bigwedge_{p_i \in v} p_i \right)$$

Exemple

Soit $\{p, q, r\}$ un ensemble de littéraux et $\Sigma = (p \wedge q) \vee (q \wedge \neg r)$. Σ est constituée de disjonction de monômes $(p \wedge q)$ et $(q \wedge \neg r)$ et on a Σ est sous forme DNF.

Propriété 1

Toute formule de la logique propositionnelle peut être réécrite vers l'une des trois formes normales. De plus, la formule originale et celle obtenue sont logiquement équivalentes.

Exemple

Soit la formule propositionnelle $\Sigma = (p \wedge q) \vee (s \wedge \neg(q \rightarrow r))$, Σ peut être écrite sous forme normale comme suit :

- NNF(Σ) $= (p \wedge q) \vee (s \wedge q \wedge \neg r)$;
- CNF(Σ) $= (p \vee s) \wedge (p \vee q) \wedge (p \vee \neg r) \wedge (q \vee s) \wedge (q) \wedge (q \vee \neg r)$;
- DNF(Σ) $= (p \wedge q) \vee (s \wedge q \wedge \neg r)$.

Dans la section suivante, nous rappelons la définition du problème de satisfiabilité, puis nous exposons les différents algorithmes pour résoudre un problème de satisfiabilité. Enfin, nous étudions comment énumérer tous les modèles d'une formule propositionnelle.

1.2 Problème SAT

Le problème de la satisfiabilité propositionnelle, ou SAT en abrégé, est un problème de décision important et bien étudié en informatique. En effet, le problème SAT est le premier problème de décision prouvé NP Complet [28], ce qui fait de lui un problème de référence dans la classe NP. Ce problème a fait et fait l'objet de nombreuses recherches depuis des décennies. Le problème SAT se définit formellement comme suit :

1.2.1 Définition du problème SAT

Définition 18 (Problème SAT)

Le problème SAT est le problème de décision qui consiste à déterminer si une formule Σ sous forme normale conjonctive CNF(Σ) admet ou non un modèle.

La procédure de détermination de la satisfiabilité d'un problème SAT est appelée un solveur ou prouveur SAT. Elle a en entrée une formule sous forme normale conjonctive (CNF). Il est généralement plus simple de modéliser en logique propositionnelle certains problèmes de décision donné suivi par l'appel à un solveur SAT pour répondre à la question de la satisfiabilité de la formule résultante de l'encodage, c'est-à-dire si la formule admet ou non un modèle. Ce processus est résumé dans le tableau suivant.

Problème :	SAT
Entrée :	Formule propositionnelle Σ
Question :	$\mathcal{M}(\Sigma) \neq \emptyset$?
Sortie :	SAT ou UNSAT

Le reste de cette section est consacré aux approches les plus utilisées pour la résolution du problème SAT dans la pratique, à savoir la procédure DPLL et la procédure CDCL en tant qu'extension de DPLL. Enfin, nous étudions les approches d'énumération de tous les modèles de ce problème.

1.2.2 Résolution d'un problème SAT

De nombreux algorithmes et méthodes ont été proposés pour résoudre les problèmes SAT. Ces algorithmes utilisent les principes de la résolution [109, 47], la réécriture [17, 114], l'énumération [29], la recherche locale [14], les diagrammes binaires de décision [5], etc. En pratique, les algorithmes basés sur le principe de résolution et les algorithmes énumératifs sont les plus utilisés. Ces approches reposent généralement sur la procédure DPLL [29]. Les récents algorithmes sont basés sur une combinaison fine de la procédure DPLL et d'une méthode basée sur le principe de résolution. Ces solveurs SAT modernes sont appelés également CDCL (Conflict Driven Terms Learning) [95].

Résolution logique

L'une des techniques les plus importantes pour la résolution du problème SAT est la règle de résolution [109]. Cette règle s'applique entre les clauses d'une formule Σ, permettant de dériver de nouvelles clauses. Sa définition formelle est la suivante :

Définition 19 (Résolvante)

Soient deux clauses ω_1 et ω_2 contenant respectivement un littéral p et son complément $\neg p$. La clause $\tau = \omega_1 \cup \omega_2 - \{p, \neg p\}$ est appelée résolvante entre ω_1 et ω_2 sur p et notée $\eta[p, \omega_1, \omega_2]$. Cette clause est obtenue à partir de l'union de ω_1 et ω_2 privée de p et $\neg p$.

Exemple

Soient $\omega_1 = (p \vee q \vee r)$ et $\omega_2 = (\neg p \vee r \vee s)$ deux clauses, nous avons $\eta[p, \omega_1, \omega_2] = (q \vee r \vee s)$.

Propriété 2

Soient Σ une formule, ω_1 et ω_2 deux clauses de Σ et la résolvante τ entre ω_1 et ω_2 sur p. Nous avons les deux propriétés suivantes :

1. $(p \vee \omega_1) \wedge (\neg p \vee \omega_2) \models \tau$;
2. $\Sigma \equiv \Sigma \cup \{\tau\}$.

L'opération de résolution est utilisée pour dériver une clause résolvante à partir de deux clauses, dont l'une de ces clauses contient un littéral et l'autre contient son complémentaire.

Définition 20 (Dérivation par résolution et réfutation)

Une clause ω est appelée une dérivation par résolution à partir d'une formule CNF Σ, s'il existe une série de clauses $\pi = [\omega_1, \omega_2, \ldots, \omega_k]$ telle que $\omega_k = \omega$ et $\forall i \leq k$, soit (1) $\omega_i \in \Sigma$, soit (2) il existe ω_m et ω_n, $\omega_i = \eta[x, \omega_n, \omega_m]$ avec $1 \leq m < i$ et $1 \leq n < i$.

Toute dérivation par résolution d'une clause vide ($\omega = \bot$) à partir de Σ est appelée réfutation. Dans ce cas, Σ est insatisfiable.

La deuxième technique importante est la subsumption. Cette règle permet de retirer des clauses d'une formule.

Définition 21 (Subsumption)

Une clause ω_1 subsume une clause ω_2 si tous les littéraux de ω_1 sont contenus dans ω_2.

Exemple

Soient $\omega_1 = (p \vee q)$ et $\omega_2 = (p \vee q \vee \neg r)$ deux clauses, la clause ω_1 subsume la clause ω_2.

Une autre technique appelée fusion est définie comme suit :

Définition 22 (Fusion)

La règle de fusion consiste à retirer toutes les occurrences d'un littéral dans une clause et ne conserver qu'une seule occurrence de ce littéral. Soit une clause ω dans une formule Σ. La fusion de $\omega = (p_1 \vee p_2 \vee \cdots \vee p_n \vee l \vee q_1 \vee q_2 \vee \cdots \vee q_m \vee l \vee r_1 \vee r_2 \vee \cdots \vee r_k \vee l)$ donne $\omega' = (p_1 \vee p_2 \vee \cdots \vee p_n \vee q_1 \vee q_2 \vee \cdots \vee q_m \vee r_1 \vee r_2 \vee \cdots \vee r_k \vee l)$.

Le principe de la résolution adopté dans les algorithmes de résolution de problèmes SAT comprend l'application des trois règles de résolution d'une formule CNF(Σ), de subsumption et de fusion. Lorsque l'algorithme obtient une clause vide durant la résolution ou lorsque aucune résolution n'est plus possible, l'algorithme s'arrête. La première situation signifie que le problème est insatisfiable car la résolution est complète pour la réfutation. En d'autres termes, si la formule Σ n'est pas satisfiable, il est garanti d'avoir une clause vide. La seconde situation signifie que le problème est satisfiable car l'algorithme a atteint une situation où toute nouvelle résolvante est subsumée par une clause existante, par conséquent la formule Σ est satisfiable.

Il existe d'autres systèmes de preuve par résolution. Certains sont plus performants, comme la résolution étendue [126], tandis que d'autres sont plus faibles, comme la résolution unitaire [32]. Une résolution unitaire est une preuve par résolution où l'une des clauses à résoudre est une clause unitaire. Bien que la résolution soit simple, elle est difficile à mettre en œuvre efficacement en raison du nombre exponentiel des clauses à générer pour atteindre la clause vide. Cela nécessite généralement un espace mémoire élevé et un temps de calcul prohibitif. Afin de réduire le nombre de clauses résolvantes produites, de nombreuses restrictions basées sur la structure de la preuve par la résolution ont été proposées. Elles sont généralement plus faciles à mettre en œuvre comme le principe de la méthode de Davis Putnam [30].

1.2.3 Algorithmes de résolution de SAT

Procédure DP

L'une des premières méthodes pour résoudre le problème SAT est proposée par Davis et Putnam en 1960 [30], nommée DP et décrite dans l'algorithme 1. Cette méthode repose sur l'utilisation de l'élimination des littéraux comme principe de résolution du problème SAT. En effet, il s'agit d'un raffinement de l'approche de Robinson. Plus précisément, l'algorithme procède de la manière suivante : étant donné une formule CNF Σ en entrée, l'algorithme vérifie d'abord si la formule n'est pas vide, c'est-à-dire ($\Sigma = \emptyset$), ou falsifiée ($\bot \in \Sigma$). Si aucune de ces deux situations ne se produit, l'algorithme sélectionne un littéral p de Σ, génère toutes les résolvantes possibles en utilisant ce littéral c'est-à-dire entre l'ensemble des clauses de Σ qui ont le littéral p et son complémentaire $\neg p$ en commun. Dans ce cas, l'algorithme simplifie la formule Σ en supprimant toutes les clauses où le littéral p et $\neg p$ apparaissent dans Σ et les remplacent par les clauses résolvantes. Le processus se répète jusqu'à ce qu'une clause vide (Σ à UNSAT) soit générée ou que la formule soit vide (Σ à SAT). À chaque itération, le sous-problème contient une variable de moins mais le nombre de clauses de la formule peut augmenter à cause de la résolution. Toutefois, la complexité de cet algorithme est exponentielle en temps et en espace.

Algorithm 1: DP Algorithm

Data: Σ formula in CNF
Result: SAT or UNSAT
while *($\mathcal{V}_\Sigma \neq \emptyset$)* **do**
    x = pickBranchingVariable($\mathcal{V}_\Sigma$);
    $\mathcal{V}_\Sigma = \mathcal{V}_\Sigma - \{x\}$;
    $\Sigma \leftarrow \Sigma - (\Sigma_x \cup \Sigma_{\neg x})$;
    $\Sigma \leftarrow \Sigma \cup \eta[x, \Sigma_x, \Sigma_{\neg x}]$;
    if *($\emptyset \in \Sigma$)* **then**
        return UNSAT;
return SAT;

Procédure DPLL

Davis et al. ont proposé ensuite l'algorithme DPLL [29]. Cet algorithme est une alternative à l'algorithme DP. Mais contrairement à ce dernier, DPLL n'utilise pas explicitement la règle de résolution et procède par division. La méthode DPLL structure l'espace de recherche sous forme d'un arbre de recherche. Ensuite, elle explore toutes les affectations possibles jusqu'à avoir une affectation satisfaisante ou conclure qu'une telle affectation n'existe pas. L'algorithme 2 décrit la procédure DPLL. Ce processus est une procédure récursive qui parcourt en profondeur d'abord l'arbre binaire où chaque nœud correspond à une interprétation partielle. Une telle interprétation est étendue par un littéral de décision et par une séquence de propagations unitaires qui consiste à identifier les affectations de variables implicites. La prochaine variable à affecter (pickBranchingVariable) est sélectionnée conformément à une heuristique (par exemple selon le nombre d'occurrences des variables, la taille des clauses, etc.). Par exemple, l'heuristique MOMS (Most Occurrences in Minimal Size) sélectionne la variable qui apparaît la plus fréquemment dans les clauses les plus courtes a été proposé par Freemann en 1996. La procédure DPLL procède par le principe de séparation une fois qu'elle a sélectionné une variable l, elle divise la formule Σ en deux sous-formules $\Sigma|_l$ et $\Sigma|_{\neg l}$. Par conséquent, les feuilles de l'arbre représentent deux cas, soit une satisfiabilité de la sous-formule ou une contradiction s'il y a au moins une clause vide dans la sous-formule, et dans ce cas la procédure DPLL effectue un retour arrière chronologique (remise en cause de la valeur de vérité de la variable de décision précédant la conséquence d'une clause vide) afin d'explorer une nouvelle branche de l'arbre.

Algorithm 2: DPLL Algorithm

Data: Σ : formula in CNF
Result: SAT or UNSAT
if $\Sigma = \{\emptyset\}$ **then**
  return SAT
if $\Sigma = \{\bot\}$ **then**
  return UNSAT
if $\exists \omega \in \Sigma \mid \omega = (l)$ **then**
  return DPLL($\Sigma|_l$);
$l \leftarrow$ pickBranchingVariable(Σ);
return (DPLL ($\Sigma|_l$) or DPLL ($\Sigma|_{\neg l}$));

L'utilisation de techniques « choix+propagation » et « retour arrière chronologique » sont suffisantes pour une exploration complète de l'espace de recherche du problème. De nouvelles règles sont mise en œuvre pour éviter l'inconvénient d'explorer un nombre important d'interprétations inutilement.

Solveur SAT moderne

Depuis les années 1990, des problématiques du monde réel tels que la vérification de modèles, la planification, etc., ont été encodées avec succès en SAT. Ces problèmes sont caractérisés par des formules propositionnelles à grande échelle avec un nombre important de clauses et de variables, rendant les solveurs DPLL incapables de résoudre ces problèmes. Pour s'attaquer à ces problèmes, de nouvelles heuristiques pour optimiser le processus de résolution ont été proposées, couplées à l'analyse des conflits donnant lieu aux solveurs CDCL. Bien que le solveur CDCL est basé à l'origine sur l'algorithme DPLL, son mécanisme est orienté vers l'apprentissage de clauses. Nous allons maintenant décrire brièvement les différents composants utilisés dans des solveurs SAT modernes.

1. *Stratégies des clauses apprises :* lorsqu'un conflit survient, la procédure DPLL effectue un retour arrière chronologique vers la décision (n-1), sans prendre en considération des raisons qui ont menées à ce conflit. Contrairement à l'analyse de conflit utilisé dans les solveurs SAT modernes qui permet d'effectuer un retour arrière non-chronologique vers un nœud de décision plus haut dans l'arbre de recherche. Le même conflit peut être rencontré plusieurs fois au cours de la recherche. Afin d'éviter cette situation, une approche consiste à ajouter une clause, qui représente un conflit, permet d'éviter de rencontrer le même conflit dans la suite de la recherche. Cette clause est appelée « clause apprise ».
2. *Stratégies de réduction de la base de clauses apprises :* l'analyse des conflits et l'apprentissage de clauses ont considérablement amélioré l'algorithme DPLL. Cependant, l'augmentation de la base des clauses apprises doit être gérée au risque de ralentir fortement le processus de propagation unitaire. Pour éviter cela, tous les solveurs SAT modernes utilisent des fonctions heuristiques [10, 12, 71] pour réduire la base des clauses apprises (Reduction Learning Clauses). Des heuristiques sont introduites pour gérer la fréquence de nettoyage des clauses apprises (fonction TimeToReduce) [35, 9, 12].
3. *Heuristiques basées sur les activités des variables :* la plupart des heuristiques de sélection de variables sont basées sur des paramètres syntaxiques. Avec le développement des solveurs SAT et l'émergence de l'analyse de conflits, une nouvelle variété d'heuristique dynamique est apparue, qui sélectionne la variable en fonction de son activité au cours du processus de résolution. Dans [147], les auteurs proposent une heuristique nommée VSIDS, qui associe des valeurs d'activité à toutes les variables. Ensuite, à chaque conflit, l'activité des variables existantes dans les clauses apprises est incrémentée. Dans le solveur Minisat [34], l'activité des variables n'est pas limitée aux variables apprises de l'analyse de conflit mais implique toutes les variables qui apparaissent dans les clauses qui provoquent un conflit.
4. *Stratégies de redémarrage* : la première stratégie de redémarrage dans les solveurs SAT a été introduite dans [52]. Par la suite, de nombreuses autres stratégies de redémarrage ont été proposées [66, 110, 57, 104]. Gomes et al. [53] ont montré expérimentalement que l'exécution d'une approche en variant l'ordre des variables sur un même problème entraîne des temps de résolution différents. Cette politique de redémarrage consiste à suspendre l'interprétation partielle en cours et à relancer la recherche en revenant à la racine de l'arbre de recherche, tout en conservant certaines informations accumulées durant les itérations précédentes telles que les clauses apprises, les activités des variables.

Procédure CDCL

CDCL est une méthode de type « décision+propagation ». L'algorithme 3 décrit ce type de solveur. L'algorithme initialise une base de clauses apprises Δ avec un ensemble vide (ligne 1), une interprétation partielle μ_p à 0 (ligne 2) et un niveau de décision dl à 0 (ligne 3). À chaque itération, une propagation unitaire est effectuée sur la formule $\Sigma \cup \Delta$ (ligne 5). Un littéral est sélectionné comme littéral de décision (ligne 7). Ce littéral est propagé selon une certaine polarité (ligne 9) au niveau de décision dl (ligne 11). Si tous les littéraux sont affectés, alors la formule est satisfaite (ligne 8) et μ_p est un modèle de Σ. Si la taille de la base de clauses apprises Δ devient volumineuse, celle-ci est réduite par une procédure de réduction de la base (reductionLearningClauses) (ligne 12). À chaque fois qu'un conflit survient par propagation (lignes $14-19$), l'algorithme vérifie si le niveau de décision est égal à 0, ce qui signifie que la formule est incohérente, auquel cas le problème est insatisfiable (ligne 14). Si ce n'est pas le cas, une clause ω' est dérivée par une procédure d'analyse de conflits (ligne 15) et un niveau de backjump *bl* est calculé à partir de la clause apprise ω' (ligne 16). Ensuite, un retour arrière est effectué à ce niveau de

Algorithm 3: CDCL Algorithm

Data: Σ : CNF formula
Result: SAT or UNSAT

1 $\Delta \leftarrow \emptyset$; /* Learning clauses */
2 $\mu_p \leftarrow 0$; /* Partial interpretation */
3 $dl \leftarrow 0$; /* Decision Level */
4 **while** *True* **do**
5 $\alpha \leftarrow$ propagate($\Sigma \cup \Delta, \mu_p$);
6 **if** $\alpha = null$ **then**
7 x $\leftarrow$ pickBranchingVariable();
8 **if** *all variables are affected* **then return** SAT;
9 $l \leftarrow$ assignToPolarity(x);
10 $dl \leftarrow dl + 1$;
11 $\mu_p \leftarrow \mu_p \cup \{l^{dl}\}$;
12 **if** *timeToReduce()* **then** reductionLearningClauses(Δ);
13 **else**
14 **if** $dl = 0$ **then return** UNSAT;
15 $\omega' \leftarrow$ AnalyzeConflit($\Sigma \cup \Delta, \mu_p, \alpha$) ; /* Learning clause */
16 $bl \leftarrow$ AnalyzeConflit($\Sigma \cup \Delta, \mu_p, \omega'$) ; /* backjump level */
17 $\Delta \leftarrow \Delta \wedge \omega'$;
18 backjump($\Sigma \cup \Delta, \mu_p, bl$);
19 **if** *restart()* **then** restart(μ_p, dl);

backjump *bl* (ligne 18). Après un certain seuil de conflits, le solveur CDCL relance la recherche depuis la racine de l'arbre (ligne 19).

1.2.4 Enumération dans SAT

Ces dernières années, l'utilisation de la satisfiabilité propositionnelle pour résoudre certains problèmes a conduit à étendre le champ d'application des solveurs SAT. Cependant, dans certaines applications il faut trouver l'ensemble de toutes les solutions possibles. Ce problème qui consiste à énumérer tous les modèles d'une formule est plus difficile que le problème SAT puisque le nombre de modèles est potentiellement exponentiel. Plusieurs algorithmes énumératifs ont été proposés pour répondre à cette problématique [93, 26, 75, 115, 83, 107, 73].

Certains algorithmes énumératifs (ou de retour arrière chronologique) déterminent l'ensemble minimal des affectations couvrant tous les modèles de la formule originale, même si certains modèles peuvent être couverts par plus d'une affectation partielle. Par conséquent, les travaux sur l'énumération de modèles se sont concentrés sur les techniques permettant de réduire la taille des affectations satisfaisantes trouvées. L'une des approches les plus utilisées en pratique pour résoudre un problème SAT est les algorithmes énumératifs reposant généralement sur la procédure DPLL. En effet, un grand nombre de domaines d'application de SAT s'appuient sur des algorithmes efficaces pour l'énumération de modèles propositionnels. Par exemple, l'énumération de modèles est utilisée pour les problèmes de vérification de modèles [73, 75, 93], compilation de connaissances et dans des solveurs hybrides [11], dont les auteurs ont proposé un solveur énumératif d'affectations propositionnelles, appelé Math-SAT en adaptant la procédure DPLL. Le tableau suivant résume un aperçu sur l'énumération de modèles.

Problème : Enumération dans SAT
Entrée : Formule propositionnelle Σ
Sortie : L'ensemble de modèles $\mathcal{M}(\Sigma)$

Une revue des principales techniques des algorithmes d'énumération des modèles d'une formule CNF est discutée dans [125]. L'utilisation des clauses bloquantes est l'une des techniques les plus utilisées dans les algorithmes d'énumération de modèles.

— *Stratégie de Clauses bloquantes :* Enumérer les modèles via les clauses bloquantes consiste à ajouter des clauses à la formule lorsqu'un modèle est trouvé pour empêcher la découverte des modèles déjà trouvés [107, 75, 83, 73]. La clause bloquante est obtenue en utilisant la négation des littéraux du modèle obtenu. Plus précisément si $\mu = \{x_1, \ldots, x_n\}$ est un modèle de la formule, alors la clause bloquante $\omega_B = (\neg x_1 \vee \ldots \vee \neg x_n)$ est ajoutée à la formule. D'autres techniques de simplification des affectations partielles sont proposées pour optimiser l'énumération de modèles [94]. Pour une présentation plus complète sur ce sujet, le lecteur pourra se référer utilement à ces ouvrages [94, 138].

L'algorithme 4 présente un algorithme de base pour obtenir tous les modèles d'une formule propositionnelle [94].

Algorithm 4: Basic model Enumeration Algorithm

```
1  Function EnumerateModels(Σ, μ) :
2      UnitPropagate(Σ, μ);
3      if ()∈ Σ then
4          return False;
5      if Σ == ∅ then
6          μ' = MinimizePartialAssignment(μ);
7          Save(μ');
8          BlockPartialAssignment(μ') ; /* Eliminate assignments that intersect
               μ' */
9          return True;
10     (x_i, v_i) = SelectBranchVariable(Σ);
11     (l_p, l_n) = (v_i == 1) ? (x_i, ¬x_i) : (¬x_i, x_i);
12     s_1 = EnumerateModels(Σ ∪ {(l_p)}, μ ∪ {(x_i, v_i)});
13     s_2 = EnumerateModels(Σ ∪ {(l_p)}, μ ∪ {(x_i, 1 − v_i)});
14     return s_1 ∨ s_2;
```

Procédure DPLL énumérative

La procédure DPLL est étendue dans l'algorithme 5 pour énumérer tous les modèles d'une formule Σ par ajout de clauses bloquantes ω_B. Seules les lignes 1 et 2 sont modifiées. Lorsqu'un modèle est trouvé, il est ajouté à l'ensemble de modèles $\mathcal{M}(\Sigma)$ (ligne 5). La fonction BlockAssignment est appelée pour générer une clause bloquante ω_B à partir du modèle. Ensuite, la formule Σ est étendue par la clause bloquante ω_B. Le solveur relance la recherche à partir de la racine de l'arbre de recherche.

Algorithm 5: DPLL_Enum : DPLL for model enumeration

Data: Σ : formula in CNF
Result: $\mathcal{M}(\Sigma)$: set of models of Σ

1 $\mathcal{M}(\Sigma) \leftarrow \emptyset$;
2 $\mu \leftarrow \emptyset$;
3 **if** $\mu \models \Sigma$ **then**
4 $\quad \omega_B \leftarrow$ BlockAssignment(μ);
5 $\quad \Sigma \leftarrow \Sigma \wedge \omega_B$;
6 $\quad$ **return** μ ; `/* new found model */`
7 **if** $\Sigma \models \bot$ **then return** $\emptyset$;
8 **if** $\exists \omega \in \Sigma \mid \omega = (l)$ **then return** DPLL($\Sigma_{|l}$);
9 $l \leftarrow$ pickBranchingVariable(Σ);
10 $\mathcal{M}_r \leftarrow DPLL\_Enum(\Sigma_{|l})$;
11 $\mathcal{M}_l \leftarrow DPLL\_Enum(\Sigma_{|\neg l})$;
12 **return** $\mathcal{M}_r \cup \mathcal{M}_l$;

Procédure CDCL énumérative

L'algorithme 6 est le pseudo-code obtenu en modifiant l'algorithme 3 afin d'énumérer tous les modèles d'une formule Σ. Seules les lignes 7-11 sont modifiées. À chaque fois qu'un modèle est produit, le modèle est ajouté à l'ensemble de modèles $\mathcal{M}(\Sigma)$. Ensuite, une clause bloquante ω_B est générée (fonction BlockAssignment) (ligne 11), ω_B est ajoutée à la formule Σ (ligne 12). Dans ce cas, la procédure effectue un retour arrière pour relancer la recherche d'un nouveau modèle (ligne 13).

Dans certains problèmes comme la fouille des motifs fréquents, le nombre de modèles est très important dans le pire des cas. Ce qui rend le nombre de clauses bloquantes à ajouter très important également. Cet ajout rend la complexité spatiale des algorithmes exponentielle. D'autre part, l'ajout de ces clauses peut ralentir considérablement la propagation unitaire connue pour être la fonction la plus coûteuse en temps. Un autre inconvénient lié aux clauses bloquantes est que chaque fois qu'une solution est trouvée, le solveur redémarre la recherche à partir de la racine de l'arbre de recherche ce qui pourra avoir un grand impact sur l'heuristique de choix de variables et la stratégie de redémarrage utilisée.

Remarque

Contrairement aux clauses apprises, les clauses bloquantes ne peuvent pas être supprimées pour éviter de redécouvrir des modèles déjà trouvés.

Dans [69], Jabbour et al. ont proposé une extension de la procédure CDCL pour l'énumération des motifs fréquents dans une séquence. Cette approche améliore le processus d'énumération en recherchant tous les modèles proches d'un modèle trouvé à une itération donnée. Plus précisément au lieu d'effectuer des retours arrière après chaque modèle, un simple backtrack est opéré au niveau de l'avant- dernière décision. De plus, la clause bloquante est restreinte aux négations des littéraux de décision.

Algorithm 6: CDCL_Enum : CDCLL for model enumeration

Data: Σ : formula in CNF
Result: $\mathcal{M}(\Sigma)$: set of models of Σ

$\Delta \leftarrow \emptyset$; /* Learning clauses */
$\mu_p \leftarrow 0$; /* Partial interpretation */
$dl \leftarrow 0$; /* Decision Level */
$\mathcal{M}(\Sigma) \leftarrow \emptyset$;
while *True* **do**
  $\alpha \leftarrow$ propagate($\Sigma \cup \Delta, \mu_p$);
  if $\alpha = null$ **then**
    x $\leftarrow$ pickBranchingVariable();
    if *all variables are affected* **then**
      $\mathcal{M}(\Sigma) \leftarrow \mathcal{M}(\Sigma) \cup \mu_p$; /* new found model */
      $\omega_B \leftarrow$ BlockAssignment(μ_p);
      $\Sigma \leftarrow \Sigma \wedge \omega_B$;
      $dl \leftarrow 0$;
    else
      $l \leftarrow$ assignToPolarity(x);
      $dl \leftarrow dl + 1$;
      $\mu_p \leftarrow \mu_p \cup \{l^{dl}\}$;
    if *timeToReduce()* **then** reductionLearningClauses(Δ);
  else
    if $dl = 0$ **then return** False;
    $\omega' \leftarrow$ AnalyzeConflit($\Sigma \cup \Delta, \mu_p, \alpha$) ; /* Learning clause */
    $bl \leftarrow$ AnalyzeConflit($\Sigma \cup \Delta, \mu_p, \omega'$) ; /* backjump level */
    $\Delta \leftarrow \Delta \cup \omega'$;
    backjump($\Sigma \cup \Delta, \mu_p, bl$);
    if *restart()* **then** restart(μ_p, dl);
return $\mathcal{M}(\Sigma)$;

1.3 Conclusion

Le problème SAT est l'un des problèmes de décision les plus connus. Par conséquent, il a fait l'objet de beaucoup de travaux qui ont donné lieu à diverses approches pour sa résolution. Le problème SAT utilise un langage simple et des méthodes pouvant résoudre de manière efficace des problèmes jugés jadis difficiles.

Dans ce chapitre, nous avons introduit les deux aspects de la logique propositionnelle, à savoir sa syntaxe et sa sémantique, puis nous avons présenté trois formes normales particulières en logique propositionnelle. Ensuite, nous avons défini le problème SAT et ses algorithmes de résolutions, à savoir les procédures DPLL et CDCL. Enfin, nous avons présenté les méthodes qui permettent d'énumérer tous les modèles d'une formule CNF.

Dans le chapitre suivant, nous présenterons le problème de la fouille des itemsets et ses deux catégories d'algorithmes à savoir les approches spécialisées et les approches déclaratives et plus particulièrement l'utilisation de la logique propositionnelle pour extraire les itemsets fréquents.

Chapitre 2

Fouille des itemsets

Sommaire

L'un des problèmes les plus étudiés dans le domaine de la fouille de données est l'extraction des itemsets. Historiquement, la motivation derrière l'extraction des itemsets et les règles d'association est venue de la nécessité d'analyser les données de transactions des supermarchés, désigné communément par le problème du panier de la ménagère (Agrawal et al.[2]) permettant de mettre en évidence la corrélation entre certains produits achetés par des clients. Dans ce chapitre, nous allons passer en revue les principes fondamentaux de la fouille des itemsets. Dans la section 2.1, nous commencerons par quelques définitions de base. Dans la section 2.2, nous présentons des approches spécialisées et des approches déclaratives pour extraire les itemsets fréquents. Des représentations condensées des itemsets fréquents sont étudiées dans la section 2.3. Le problème de la fouille des itemsets fréquents est étendu à d'autres variantes dans la section 2.4.

2.1 Définitions

Nous considérons un ensemble fini et non vide d'items Ω.

Définition 23

Un itemset I sur Ω est un sous-ensemble de Ω, c'est-à-dire, $I \subseteq \Omega$.

Le cardinal d'un itemset est noté par $|I|$.

Définition 24

Un itemset I est appelé un k-itemset s'il est constitué de k items, c'est-à-dire $|I| = k$.

Nous utilisons les lettres alphabétiques minuscules a, b, c, etc... pour désigner les items et les lettres alphabétiques majuscules I, J, K,etc... pour désigner les itemsets. Nous notons l'ensemble de tous les itemsets sur Ω par 2^{Ω}.

Exemple

L'itemset {a,b,d} est un 3-itemset car il est constitué de trois items a, b et d.

Définition 25

Une transaction T_i est une paire (i, I) formée d'un identifiant de transaction i tel que $1 \leq i \leq m$ et d'un itemset I.

Notons bien que chaque identifiant de transaction correspond à un itemset unique. Pour une transaction $T_i = (i, I)$, la taille de T_i est notée par $|T_i| = |I|$.

Définition 26

Une base de données transactionnelle $\mathcal{D} = \{T_1, T_2, \ldots, T_m\}$ est un ensemble de m transactions.

La table 2.1 représente un exemple d'une base de données transactionnelle $\mathcal{D}_1 = \{(1, \{c, e, f, g\}), (2, \{b, d, e, f, g, h\}), (3, \{a, b, d\}), (4, \{a, b, d, f, h\}), (5, \{b, c, e, f, g, h\}), (6, \{c, g\}), (7, \{a, b, d, h\}), (8, \{c, e, g\}), (9, \{a, b, d\}), (10, \{b, c, d, e, f, g, h\})\}$ de 10 transactions définie sur l'ensemble de 8 items $\Omega = \{a, b, c, d, e, f, g, h\}$. Notons que la base de données des transactions peut être représentée comme une matrice booléenne, où chaque transaction est un vecteur booléen de tous les items.

Tid	Transaction
1	$\{c, e, f, g\}$
2	$\{b, d, e, f, g, h\}$
3	$\{a, b, d\}$
4	$\{a, b, d, f, h\}$
5	$\{b, c, e, f, g, h\}$
6	$\{c, g\}$
7	$\{a, b, d, h\}$
8	$\{c, e, g\}$
9	$\{a, b, d\}$
10	$\{b, c, d, e, f, g, h\}$

(a) Représentation horizontale de $\mathcal{D}_1$

Tid	a	b	c	d	e	f	g	h
1	0	0	1	0	1	1	1	0
2	0	1	0	1	1	1	1	1
3	1	1	0	1	0	0	0	0
4	1	1	0	1	0	1	0	1
5	0	1	1	0	1	1	1	1
6	0	0	1	0	0	0	1	0
7	1	1	0	1	0	0	0	1
8	0	0	1	0	1	0	1	0
9	1	1	0	1	0	0	0	0
10	0	1	1	1	1	1	1	1

(b) Représentation booléenne de $\mathcal{D}_1$

TABLE 2.1 – Exemple d'une base transactionnelle $\mathcal{D}_1$.

Étant donné une base de transactions $\mathcal{D}$ et un itemset I, la couverture de I dans $\mathcal{D}$, notée $Cov(I, \mathcal{D})$, est définie comme suit :

Définition 27

La couverture d'un itemset I comprend toutes les transactions qui contiennent l'itemset I, c'est-à-dire $Cov(I, \mathcal{D}) = \{i \in \mathbb{N} \mid (i, J) \in \mathcal{D} \; et \; I \subseteq J\}$

La définition suivante décrit le support de I dans la base de données $\mathcal{D}$, noté $Supp(I, \mathcal{D})$.

Définition 28

Le support d'un itemset I est la taille de la couverture $Cov(I, \mathcal{D})$, c'est-à-dire $Supp(I, \mathcal{D}) = |Cov(I, \mathcal{D})|$.

Exemple

Dans la base transactionnelle $\mathcal{D}_1$, nous avons $Cov(\{b, d, e\}, \mathcal{D}) = \{2, 10\}$ et $Supp(\{b, d, e\}, \mathcal{D}) = |\{2, 10\}| = 2$.

Soit $\mathcal{D}$ une base de transactions sur Ω et λ un seuil minimum de support. Le problème de l'extraction des itemsets fréquents est formellement définie de la manière suivante :

Définition 29

Le problème de la fouille des itemsets, noté $\mathcal{FIM}(\mathcal{D}, \lambda)$, consiste à énumérer tous les itemsets fréquents, c'est-à-dire

$$\mathcal{FIM}(\mathcal{D}, \lambda) = \{I \subseteq \Omega \mid Supp(I, \mathcal{D}) \geq \lambda\}$$

La table 2.2 illustre l'ensemble des itemsets fréquents trouvés dans la base représentée dans la table 2.1. Le seuil minimum de support spécifié λ est égal à 5.

Itemset	Support
$\{b\}$	7
$\{c\}$	5
$\{d\}$	6
$\{e\}$	5
$\{f\}$	5
$\{g\}$	6
$\{h\}$	5
$\{c, g\}$	5
$\{e, g\}$	5
$\{b, h\}$	5
$\{b, d\}$	6

TABLE 2.2 – Itemsets fréquents associés à leurs supports.

Propriété 3

Soient $\mathcal{D}$ une base de données de transactions et λ un seuil minimum de support. Soient I et J deux itemsets. Si $I \subseteq J$ alors $Supp(I, \mathcal{D}) \geq Supp(J, \mathcal{D})$. Par conséquent, si $J \in \mathcal{FIM}(\mathcal{D}, \lambda)$ alors $I \in \mathcal{FIM}(\mathcal{D}, \lambda)$.

La propriété 3 signifie que tous les sous-itemsets d'un itemset fréquent sont également fréquents. Par conséquent, la contrainte de support est anti-monotone. Cette propriété est largement utilisée pour élaguer l'espace de recherche des algorithmes d'extraction des itemsets fréquents.

2.2 Approches de la fouille des itemsets fréquents

De nombreux algorithmes ont été proposés pour résoudre le problème de trouver des itemsets fréquents. Parmi les algorithmes de base, nous pouvons citer Apriori [3], Fp-Growth [60], Eclat [145]. Ces algorithmes utilisent une base de transactions $\mathcal{D}$ et un seuil minimum de support λ en entrée et fournissent l'ensemble des itemsets fréquents en sortie. D'une part, le point commun de tous les algorithmes d'extraction des itemsets fréquents est qu'ils produisent les mêmes résultats pour une base transactionnelle et une valeur de seuil minimum de support. D'autre part, la différence entre ces algorithmes pour trouver des itemsets fréquentes est la stratégie de recherche utilisée, en largeur d'abord ou en profondeur d'abord, qui détermine la manière de générer l'ensemble des itemsets suivants et la manière d'explorer l'espace de recherche. La seconde différence concerne la forme de la base de transactions utilisée, qu'elle soit horizontale, verticale ou booléenne. Cela affecte la façon dont le support des itemsets est calculé [40].

En général, il existe deux grandes catégories de méthodes de la fouille des itemsets, comme le montre la figure 2.1. La première implique les méthodes spécialisées pour extraire des itemsets intéressants, comme Apriori et FP-growth. Cependant, afin de s'adapter aux nouvelles contraintes des utilisateurs, un grand nombre de ces méthodes utilisent des techniques qui nécessitent de nouvelles implémentations. Pour résoudre ce problème, des travaux ont récemment utilisé des approches déclaratives, à savoir la programmation par contraintes (CP) et la satisfiabilité propositionnelle (SAT) pour bénéficier de leur flexibilité et de leur généricité vis-à-vis des nouvelles contraintes des utilisateurs.

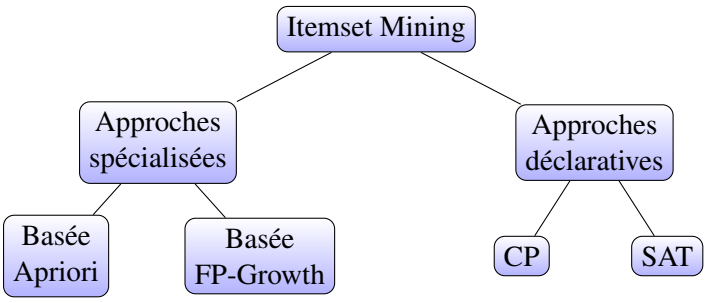


FIGURE 2.1 – Approches de la fouille des itemsets.

2.2.1 Approches spécialisées

Dans la catégorie des approches spécialisées, nous présentons les algorithmes de base pour résoudre le problème d'extraction des itemsets fréquents. Ce problème est un problème d'énumération, qui vise à

énumérer tous les itemsets en satisfaisant la contrainte de support minimum définie par l'utilisateur. Une approche naïve consiste à générer tous les itemsets possibles, puis à ne conserver que les itemsets qui vérifient le seuil minimum de support. Cependant, étant donné que le nombre des itemsets possibles est de $2^m - 1$, pour m items différents, une telle stratégie est inefficace. Afin d'éviter une exploration entière de l'espace de recherche, des algorithmes s'appuient sur la propriété d'anti-monotonie, c'est-à-dire si un itemset est non fréquent, alors tous ses sur-ensembles sont également non fréquents. Dans cette catégorie, nous pouvons se référer au premier algorithme proposé « Apriori ».

Apriori : algorithme horizontal de recherche en largeur

Apriori est un algorithme itératif proposé par Agrawal et al. [3] pour trouver des itemsets fréquents par niveau. Le pseudo-code de l'algorithme Apriori est donné dans Algorithme 7. Apriori analyse d'abord la base de transactions pour déterminer le support de chaque item (ligne 1). Il identifie ensuite l'ensemble de tous les items fréquents, désigné par F_1 (ligne 2). Ensuite, Apriori effectue une stratégie de recherche en largeur d'abord pour trouver un ensemble fréquent plus large (ligne 4 à 10). Durant le processus de recherche, Apriori utilise les itemsets fréquents du niveau $k-1$ (représenté par F_{k-1}) pour générer l'ensemble des k-itemsets candidats (spécifiés par C_k). Cela se fait en combinant des paires de k-itemsets, ces k-itemsets partagent tous les items sauf un (ligne 5). Pour les 1-itemsets fréquents $\{a\}, \{b\}$ et $\{d\}$, Apriori combine ces itemsets pour générer les 2-itemsets candidats : $\{a, b\}, \{a, d\}, \{b, d\}$.

Après avoir généré des k-itemsets candidats, l'algorithme vérifie si les $k-1$-sous-itemsets de chaque candidat sont fréquents afin de ne pas violer la propriété d'anti-monotonie. Si l'un des $k-1$-sous-itemsets du k-itemset candidat I est non fréquent, ce dernier ne peut pas être fréquent, et il est donc supprimé de l'ensemble des candidats C_k (ligne 6). Pour déterminer le support de tous les itemsets candidats restants dans C_k, l'algorithme analyse ensuite la base de transactions et calcule le support de chaque candidat (ligne 7). Si chaque candidat a au moins un support supérieur ou égal au seuil minimum de support λ défini par l'utilisateur, il est ajouté à l'ensemble des k-itemsets fréquents F_k (ligne 8). Ce processus est itéré et les k-itemsets servent à leurs tours pour la génération de $k+1$-itemsets candidats. Si aucun autre candidat n'est trouvé, l'algorithme s'arrêtera et retourne à l'utilisateur tous les itemsets fréquents (ligne 11).

Algorithm 7: *Apriori* Algorithm

Data: $\mathcal{D}$: a horizontale transaction database, λ : a user-specified threshold
Result: The set of all frequent itemsets

1 Scan the database to calculate the support of all items in Ω ;
2 $F_1 \leftarrow \{a | a \in \Omega \wedge Supp(\{a\}, \mathcal{D}) \geq \lambda\}$; `/* F1 : frequent 1-itemsets */`
3 $k \leftarrow 2$;
4 **while** $(F_k \neq \emptyset)$ **do**
5 $C_k \leftarrow CandidateGeneration(F_{k-1})$; `/* Ck : candidate k-itemset */`
6 Remove each candidate $I \in C_k$ that contains a $(k-1)$-itemset that is not in F_{k-1} ;
7 Scan the database to calculate the support of each candidate $I \in C_k$;
8 $F_k \leftarrow \{I | I \in C_k \wedge Supp(I, \mathcal{D}) \geq \lambda\}$; `/* Fk : frequent k-itemsets */`
9 $k \leftarrow k + 1$;
10 **return** $(\bigcup_{k=1...k} F_k)$;

Une des limites de l'algorithme Apriori est le coût des balayages multiples nécessaire de la base de données pour calculer le support des itemsets candidats. La deuxième limite est causée par l'étape de génération de tous les candidats possibles dans l'algorithme Apriori. De nombreux itemsets candidats

n'existent pas dans la base de transactions. La complexité en temps de l'algorithme Apriori est O(d^2m) tel que d est le nombre d'items et m est le nombre de transactions[61]. Des recherches approfondies ont été menées pour améliorer ou étendre Apriori. Par exemple, dans [97] une technique de hachage est utilisée pour calculer efficacement le support des itemsets candidats, et une technique de partitionnement est proposée dans [111].

Eclat : algorithme vertical de recherche en profondeur

Eclat est un algorithme proposé par Zaki dans [145] pour l'extraction des itemsets fréquents en explorant l'espace de recherche en profondeur d'abord. Cet algorithme est une amélioration des limites de l'algorithme Apriori. L'idée de cet algorithme est de ne pas maintenir de nombreux itemsets en mémoire afin d'éviter la surcharge de la mémoire. Le pseudo-code d'Eclat est présenté dans l'Algorithme 8. Il utilise en entrée une base verticale sous forme d'un couple (Item(x), $tid(x)$), où $tid(x)$ est une liste d'identifiants des transactions dont l'item x apparaît. La représentation verticale de la base de données présentée dans la table 2.1 est indiquée dans la table 2.3.

Item (x)	$tid(x)$
a	$\{T_3, T_4, T_7, T_9\}$
b	$\{T_2, T_3, T_4, T_5, T_7, T_9, T_{10}\}$
c	$\{T_1, T_5, T_6, T_8, T_{10}\}$
d	$\{T_2, T_3, T_4, T_7, T_9, T_{10}\}$
e	$\{T_1, T_2, T_5, T_8, T_{10}\}$
f	$\{T_1, T_2, T_4, T_5, T_{10}\}$
g	$\{T_1, T_2, T_5, T_6, T_8, T_{10}\}$
h	$\{T_2, T_4, T_5, T_7, T_{10}\}$

TABLE 2.3 – Représentation verticale de la base de données $\mathcal{D}_1$.

L'algorithme Eclat prend en entrée une base de données verticale $\mathcal{D}$ (items et leurs listes tid) et un seuil minimum de support λ. Il trouve les itemsets fréquents en se basant sur le concept de classes d'équivalence et la recherche en profondeur d'abord (ligne 4). Deux k-itemset font partie de la même classe d'équivalence, s'ils partagent $(k-1)$-itemset comme préfixe. Par exemple les deux itemsets $\{a, b, c\}$ et $\{a, b, d\}$ sont de la même classe d'équivalence. Pour déterminer le support d'un itemset $I \cup J$ sans scanner la base de transactions originale $\mathcal{D}$ il suffit de calculer $|tid(I \cup J) = tid(I) \cap tid(J)$. Par conséquent, Eclat effectue une opération d'intersection entre toutes les listes $tid(I)$ et $tid(J)$ (ligne 5), c'est-à-dire $tid(I) \cap tid(J)$. Le support de l'itemset $I \cup J$ est déduit $Supp(I \cup J, \mathcal{D}) = |tid(I \cup J)|$. Par exemple, soit la base de transactions présentée dans la table 2.3 et l'itemset $\{a, d\}$, sa liste des $tid(\{a, d\})$ peut être obtenue comme suit $tid(\{a, d\}) = tid(a) \cap tid(d) = \{T_3, T_4, T_9\}$ et on peut donc en déduire que $Supp(\{a, d\}, \mathcal{D}) = |\{T_3, T_4, T_9\}| = 3$. Si le support $Supp(I, \mathcal{D})$ est supérieur ou égal à λ (ligne 6) alors l'itemset I est ajouté à l'ensemble des itemsets fréquents E. Ensuite, l'algorithme Eclat est appelé récursivement avec l'ensemble E pour explorer toutes les extensions de I (ligne 8).

L'algorithme Eclat présente d'importants inconvénients. Tout d'abord, Eclat peut passer du temps à examiner des itemsets qui n'existent pas dans la base de données car il génère des candidats sans scanner la base de données. Deuxièmement, bien que les listes tid soient précieuses, elles peuvent occuper beaucoup d'espace mémoire, en particulier pour les bases de données denses où tous les items apparaissent dans presque toutes les transactions.

Algorithm 8: Eclat (**E**quivalence **CLA**ss **T**ransformation) Algorithm

Data: $\mathcal{D}$: a set of itemsets with their tidsets, λ : a user-specified threshold
Result: The set of all frequent itemsets

```
for (itemset I ∈ D such that |tid(I)| ≥ λ) do
    Output I ;                                   /* I is a frequent itemset */
    E = ∅ ;          /* frequent itemsets that are extensions of I */
    for (itemset J ∈ D sharing all but the last item with I) do
        tid(I ∪ J) = tid(I) ∩ tid(J) ;      /* calculate the tidset of I ∪ J */
        if |tid(I ∪ J)| ≥ λ then
            E = E ∪ {I ∪ J} ;  /* add I ∪ J to frequent extensions of I */
        Eclat(E, λ) ;                            /* recursive call using E */
```

FP-growth : algorithme de croissance de motifs

Les algorithmes de croissance de motifs tels que FP-growth [60] ont introduit des avancées significatives pour résoudre les limitations des algorithmes comme Apriori et Eclat. L'idée de base des algorithmes de croissance de motifs consiste à analyser une base de données et de découvrir que les itemsets fréquents en évitant la génération des itemsets candidats qui n'existent pas dans la base de données.

FP-growth utilise une structure de données qui compresse la base de données en un arbre de motifs appelé FP-tree (Frequent Pattern tree), où chaque branche de l'arbre représente une ou de multiples transactions de la base originale. L'algorithme FP-growth effectue quatre phases pour extraire les itemsets fréquents. Les deux premières phases de FP-growth sont présentés dans l'algorithme 9 et les deux dernières phases sont présentés dans l'algorithme 10.

Algorithm 9: FP-tree construction algorithm

Data: $\mathcal{D}$: a transaction database, λ : a user-specified threshold
Result: FP-tree : frequent pattern tree

```
Scan the transaction database D once ;
Collect the set of frequent items F and their supports ;
Sort F in support descending order as L, the list of frequent items ;
Create the root of an FP-tree, T , and label it as "null" ;
for each transaction T ∈ D do
    Select and sort the frequent items in Trans according to the order of L ;
    Let the sorted frequent item list in Trans be [p|P ], where p is the first element and P is the
      remaining list ;
    Call insert_tree([p|P], T ) ;
if P ∉ ∅ then call insert_tree(P, N) recursively;
End algorithm
Function insert_tree([p|P], T) :
    if T has a child N such that N.item-name = p.item-name then
        increment N's count by 1 ;
    else
        create a new node N, and let its count be 1 ;
        its parent link be linked to T ;
        its node-link be linked to the nodes with the same item-name via the node-link structure ;
    End function
```

Algorithm 10: FP-growth algorithm

Data: FP-tree : frequent pattern tree, λ : a user-specified threshold
Result: The set of all frequent itemsets

```
if Tree contains a single path P then
    for each combination β of the nodes in the path P do
        generate pattern α ∪ β with support = minimum support of nodes in β;
else
    for each combination a_i in the header of Tree do
        generate pattern β = a_i ∪ α with support = a_i.support;
        construct β's conditional pattern base and then β's conditional FP-tree Tree_β;
        if Tree_β ≠ ∅ then
            call FP-growth(Tree_β, β);
```

— *Extraction des items fréquents* : la phase initiale de FP-growth consiste à analyser la base de données et à extraire tous les 1-itemsets fréquents F_1 et leur support dans la base de données. Puis, l'ensemble des 1-itemsets fréquents est trié par ordre décroissant de support, les items ayant le même support sont ordonnés par l'ordre alphabétique. Par exemple, pour la base de données $\mathcal{D}_1$ et $\lambda = 5$, le résultat de la phase initiale est $F_1 = \{\{b\} : 7, \{d\} : 6, \{g\} : 6, \{c\} : 5, \{e\} : 5, \{f\} : 5, \{h\} : 5\}$.

— *Construction d'un arbre FP* : dans la seconde phase un arbre FP est construit. La première étape consiste à créer la racine de l'arbre (représentée par null). La seconde étape consiste à effectuer une deuxième analyse de la base de transactions $\mathcal{D}$ pour identifier les branches de l'arbre qui correspondent aux transactions. Avant de créer une branche, la transaction courante T_i est réordonnée selon les items fréquents F_1. Chaque item d'une transaction est représenté par un nœud dans la branche. Si une nouvelle branche à construire partage un ou des nœuds préfixes communs avec une branche existante dans l'arbre, le compteur de ces nœuds est incrémenté et les nœuds distincts sont ajoutés au dernier nœud préfixe. Nous utilisons la puissance $\{item\}^{cp}$ pour représenter le compteur cp d'un item. Enfin, les nœuds ayant le même item sont reliés par des arêtes orientées en pointillés. Par exemple, la transaction $T_1 = (1, \{c, e, f, g\})$ est réordonnée en $T_1 = (1, \{g, c, e, f\})$, cette transaction contient quatre items $\{g\} : 6$, $\{c\} : 5$, $\{e\} : 5$, $\{f\} : 5$. Alors, une branche $(null \rightarrow \{g\}^1 \rightarrow \{c\}^1 \rightarrow \{e\}^1 \rightarrow \{f\}^1)$ est créée dont $\{g\}$ est lié à la racine, $\{c\}$ est lié à $\{g\}$, $\{e\}$ lié à $\{c\}$ et $\{f\}$ est lié à $\{e\}$. La deuxième transaction $T_2 = (2, \{b, d, e, f, g, h\})$ est réordonnée en $T_2 = (1, \{b, d, g, e, f, h\})$, cette transaction contient six items $\{b\} : 7$, $\{d\} : 6$, $\{g\} : 6$, $\{e\} : 5$, $\{f\} : 5$ et $\{h\} : 5$. Alors, une branche $(null \rightarrow \{b\}^1 \rightarrow \{d\}^1 \rightarrow \{g\}^1 \rightarrow \{e\}^1 \rightarrow \{f\}^1 \rightarrow \{h\}^1)$ est créée. Nous remarquons comme cette branche n'a aucun préfixe commun avec l'arbre, cette branche sera liée à la racine. Considérons maintenant la transaction $T_3 = (3, \{b, d\})$ (après le réordonnement). Les nœuds $\{b\}, \{d\}$ sont des nœuds préfixes communs dans l'arbre FP. Par conséquent, leurs compteurs sont incrémentés en $(null \rightarrow \{b\}^2 \rightarrow \{d\}^2)$ sans créer une nouvelle branche. La transaction réordonnée $T_4 = (4, \{b, d, f, h\})$ est représentée par une branche $(null \rightarrow \{b\}^3 \rightarrow \{d\}^3 \rightarrow \{f\}^1 \rightarrow \{h\}^1)$. Cette branche est obtenue en incrémentant les compteurs des nœuds préfixes $\{b\}^3$ et $\{d\}^3$, puis en créant le nœud $\{f\}^1$ et en le liant au dernier nœud préfixe $\{d\}^3$, ensuite en créant le nœud $\{h\}^1$ et le lier au nœud $\{f\}^1$. Ces opérations sont effectuées pour toutes les transactions restantes de la base de données. La figure 2.2 illustre un arbre de motifs FP construit à partir de $\mathcal{D}_1$ et un seuil minimum de support $\lambda = 5$.

— *Construction d'un arbre conditionnel pour chaque item* : la troisième phase consiste à construire une base de motifs conditionnels puis un arbre de motifs conditionnels pour chaque item fré-

quent (ligne 7). Ce processus commence par les items fréquents dans l'ordre croissant. Une base de motifs conditionnels d'un item a est formée à partir de l'arbre FP, où chaque transaction est l'ensemble des items préfixes de l'items a depuis une branche dans l'arbre FP. Ensuite, un arbre de motifs conditionnels est construit en utilisant la base de motifs conditionnels. Par exemple, l'item h apparaît dans cinq branches de l'arbre FP, et donc la base de motifs conditionnels du suffixe h contient cinq transactions : $\{b, g, c, e, f\} : 1, \{b, d, g, c, e, f\} : 1, \{b, d, g, e, f\} : 1, \{b, d, f\} : 1, \{b, d\} : 1$. L'arbre de motifs conditionnels ne contient que le nœud $\{b\}^5$ et les items $\{d\}, \{g\}, \{c\}$ et $\{e\}$ ne sont pas considérés car leurs supports sont inférieurs au seuil minimum de support $\lambda = 5$.

— *Extraction des itemsets en utilisant l'arbre de motifs* : la dernière phase consiste à extraire les itemsets fréquents en fouillant l'arbre de motifs conditionnels. Les itemsets fréquents sont générés en combinant l'item suffixe avec les nœuds des branches ayant un support supérieur ou égal à λ. L'arbre de motifs conditionnel de l'item h permet de générer l'itemset fréquent $\{b, h\}$.

La complexité de FP-Growth dépend fortement de la recherche des itemsets dans l'arbre FP-Tree, ce qui dépend de la profondeur de l'arbre. La complexité de l'algorithme FP-growth est O(le nombre d'items fréquents $\times$ la profondeur maximale de l'arbre).

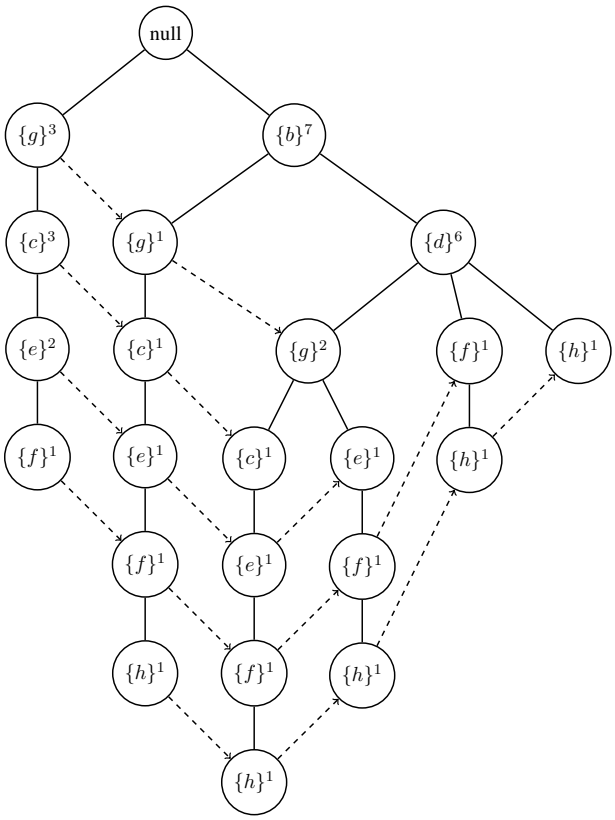


FIGURE 2.2 – Arbre FP de la base de données $\mathcal{D}_1$ avec $\lambda = 5$.

2.2.2 Approches déclaratives

Dans des travaux récents, deux approches déclaratives basées sur les contraintes sont utilisées pour résoudre les problèmes de la fouille des itemsets, à savoir la programmation par contraintes (CP) [105, 81, 112, 56] et la satisfiabilité propositionnelle (SAT) [67, 70, 31]. Dans cette section, nous présentons

un modèle basé sur la programmation par contraintes et un encodage basé sur la satisfiabilité propositionnelle pour l'extraction des itemsets fréquents.

Programmation par contraintes pour l'extraction des itemsets

Raedt et al. [105] ont formulé le modèle initial pour le problème d'extraction des itemsets fréquents basé sur la programmation par contraintes. Avant d'introduire ce modèle, nous rappelons que la programmation par contraintes est un paradigme déclaratif, qui permet de modéliser un problème sous forme de contraintes. Il s'agit d'un problème de satisfaction de contraintes (CSP), défini comme suit :

Définition 30 (CSP)

- Un ensemble fini de variables $\mathcal{V}$;
- Un ensemble des domaines D de variables $\mathcal{V}$, où chaque variable $v_a \in \mathcal{V}$ est associée à un ensemble fini de valeurs possibles $\mathcal{D}om(v_a)$;
- Un ensemble fini de contraintes $\mathcal{C}$, où une contrainte est définie sur un sous-ensemble de $\mathcal{V}$.

La résolution du problème de satisfaction des contraintes (CSP) consiste à trouver une affectation en satisfaisant toutes les contraintes. Différents solveurs ont été proposés pour résoudre le problème de satisfaction des contraintes et énumérer toutes les affectations des variables satisfaisant l'ensemble de contraintes.

Dans l'approche proposée dans [105], les items et les transactions sont représentés par des variables. Pour représenter un itemset I, une variable booléenne v_a est associée à chaque item a, $v_a = 1$ si l'item $a \in I$, sinon $v_a = 0$. Pour exprimer la couverture de l'itemset I, une variable booléenne t_i est associée à chaque transaction T_i, $t_i = 1$ si l'itemset $I \subseteq T_i$, sinon $t_i = 0$.

Soit un itemset I et une base de transactions $\mathcal{D}$ en représentation binaire, la couverture d'un itemset $Cov(I,\mathcal{D})$ est exprimée dans la contrainte suivante :

$$\forall\, T_i \in \mathcal{D} : t_i = 1 \leftrightarrow \sum_{a \in \Omega} v_a(1 - \mathcal{D}_{ia}) = 0 \tag{2.1}$$

où $v_a, t_i \in \{0,1\}$ et $v_a = 1$ si et seulement $a \in I$ et $t_i = 1$ si et seulement si $T_i \in Cov(I,\mathcal{D})$. La contrainte 2.1 garantit que la variable t_i est mise à 0 si l'itemset I est inclus T_i. Autrement dit, si un item $a \in I$ est absent de T_i, alors $v_a(1 - \mathcal{D}_{ia}) = 1$, par conséquent la contrainte $\sum_{a \in I} v_a(1 - \mathcal{D}_{ia}) = 0$ est violée, dans ce cas t_i est mise à 0.

Le support $Supp(I,\mathcal{D})$ est obtenu en comptant le nombre de transactions T_i pour lesquelles $t_i = 1$. La contrainte suivante représente la contrainte de support.

$$\sum_{T_i \in \mathcal{D}} t_i \geq \lambda \tag{2.2}$$

Où $t_i \in \{0,1\}$ et $t_i = 1$ si et seulement si $T_i \in Cov(I,\mathcal{D})$.

Le modèle du problème d'extraction des itemsets fréquents est formé de la contrainte de couverture (2.1) et de la contrainte de support (2.2).

Exemple

La modélisation du problème de la fouille des itemsets fréquents dans la base $\mathcal{D}_1$ avec $\lambda = 5$ en CP correspond aux contraintes suivantes :
$t_1 = 1 \leftrightarrow v_a + v_b + v_d + v_h = 0$
$t_2 = 1 \leftrightarrow v_a + v_c = 0$
$t_3 = 1 \leftrightarrow v_c + v_e + v_f + v_g + v_h = 0$
$t_4 = 1 \leftrightarrow v_c + v_e + v_g = 0$
$t_5 = 1 \leftrightarrow v_a + v_d = 0$
$t_6 = 1 \leftrightarrow v_a + v_b + v_d + v_e + v_f + v_h = 0$
$t_7 = 1 \leftrightarrow v_d + v_e + v_f + v_g = 0$
$t_8 = 1 \leftrightarrow v_a + v_b + v_d + v_f + v_h = 0$
$t_9 = 1 \leftrightarrow v_c + v_e + v_f + v_g + v_f = 0$
$t_{10} = 1 \leftrightarrow v_a = 0$
$t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + t_8 + t_9 + t_{10} \geq 5$

Une amélioration du modèle est apportée en utilisant les contraintes réifiées permettant de réduire la taille de l'arbre de recherche.

Définition 31

Une contrainte réifiée lie une valeur de vérité d'une contrainte C à une variable booléenne x : $x \leftrightarrow C$.

La version réifiée de la contrainte de support est exprimée sur chaque item par la contrainte suivante :

$$\forall a \in \Omega : v_a = 1 \rightarrow \sum_{T_i \in \mathcal{D}} t_i \mathcal{D}_{ia} \geq \lambda \tag{2.3}$$

La contrainte (2.3) garantit si un item n'est pas couvert par λ transactions alors cet item ne sera pas inclus dans l'itemset candidat.

Le modèle amélioré est formé de la contrainte de couverture (2.1) et de la contrainte réifiée de support (2.3).

Exemple

Reprenons l'exemple précédent. La contrainte $t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + t_8 + t_9 + t_{10} \geq 5$ est substituée par l'ensemble de contraintes suivant :
$v_a = 1 \rightarrow t_3 + t_4 + t_7 + t_9 \geq 5$
$v_b = 1 \rightarrow t_2 + t_3 + t_4 + t_5 + t_7 + t_9 + t_{10} \geq 5$
$v_c = 1 \rightarrow t_2 + t_3 + t_4 + t_5 + t_7 + t_9 + t_{10} \geq 5$
$v_d = 1 \rightarrow t_2 + t_3 + t_4 + t_7 + t_9 + t_{10} \geq 5$
$v_e = 1 \rightarrow t_1 + t_2 + t_5 + t_8 + t_{10} \geq 5$
$v_f = 1 \rightarrow t_1 + t_2 + t_4 + t_5 + t_{10} \geq 5$
$v_g = 1 \rightarrow t_1 + t_2 + t_5 + t_6 + t_8 + t_10 \geq 5$
$v_h = 1 \rightarrow t_2 + t_4 + t_5 + t_7 + t_{10} \geq 5$

Ce modèle est équivalent au modèle initial et les mêmes itemsets fréquents que le modèle initial peuvent être trouvés. En pratique, ce modèle génère un nombre important de contraintes $(n+m)$ pour l'extraction des itemsets à partir de grandes bases de données avec n items distincts et m transactions.

Encodage SAT pour l'extraction des itemsets

Dans cette section, nous présentons un encodage SAT proposé par Jabbour et al. [67] pour le problème de la fouille des itemsets fréquents. Fondamentalement, pour encoder le problème de l'extraction des itemsets dans SAT, et de manière similaire à l'encodage en CSP, il faut introduire un ensemble de variables et un ensemble de contraintes sur ces variables. Pour représenter un itemset candidat I, une variable propositionnelle p_a est associée à chaque item a tel que $p_a = vrai$ si l'itemset I contient a. Pour la couverture de I, de nouvelles variables q_i sont également introduites pour chaque identifiant de transaction $i \in \{1 \dots m\}$. De plus, un ensemble de contraintes sont introduites impliquant ces variables. Tout d'abord, la contrainte permettant de capturer la couverture d'un itemset est comme suit :

$$\bigwedge_{i=1}^{m} (\neg q_i \leftrightarrow \bigvee_{a \notin T_i} p_a) \tag{2.4}$$

Cette contrainte exprime que q_i est fausse si et seulement si l'itemset candidat n'est pas dans la transaction T_i, c'est-à-dire il y a au moins un item a dans l'itemset candidat qui n'appartient pas à la transaction.

Les auteurs ont proposé la contrainte de cardinalité suivante pour exprimer que seuls les k-itemsets sont autorisés, tel que k est égal à un seuil minimum de taille noté min :

$$\sum_{a \in \Omega} p_a \geq min \tag{2.5}$$

Enfin, la contrainte de fréquence, peut être simplement exprimée comme suit :

$$\sum_{i=1}^{m} q_i \geq \lambda \tag{2.6}$$

La tâche de l'extraction des itemsets fréquents correspond à l'énumération de modèles de la formule CNF, notée $\Sigma_{\mathcal{D},\lambda}$. Cette formule est formée essentiellement de la conjonction de contraintes (2.4) et (2.6).

Exemple

Reconsidérons la base de transactions illustrée dans 2.1. Le problème de la fouille des itemsets fréquents consiste à énumérer les modèles de la formule suivante :

$$\begin{aligned}
&\neg q_1 \leftrightarrow (p_a \vee p_b \vee p_d \vee p_h)\\
&\neg q_2 \leftrightarrow (p_a \vee p_c)\\
&\neg q_3 \leftrightarrow (p_c \vee p_e \vee p_f \vee p_g \vee p_h)\\
&\neg q_4 \leftrightarrow (p_c \vee p_e \vee p_g)\\
&\neg q_5 \leftrightarrow (p_a \vee p_d)\\
&\neg q_6 \leftrightarrow (p_a \vee p_b \vee p_d \vee p_e \vee p_f \vee p_h)\\
&\neg q_7 \leftrightarrow (p_d \vee p_e \vee p_f \vee p_g)\\
&\neg q_8 \leftrightarrow (p_a \vee p_b \vee p_d \vee p_f \vee p_h)\\
&\neg q_9 \leftrightarrow (p_c \vee p_e \vee p_f \vee p_g \vee p_h)\\
&\neg q_{10} \leftrightarrow (p_a)\\
&q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + q_7 + q_8 + q_9 + q_{10} \geq \lambda
\end{aligned}$$

2.3 Représentations condensées

L'une des limitations importantes du problème de la fouille des itemsets fréquents est le nombre des itemsets fréquents qui dépend fortement du seuil minimum de support défini par l'utilisateur. Par conséquent, il est difficile d'avoir une analyse pertinente face à un grand nombre d'itemsets. De plus, si les itemsets sont fréquents, les sous-ensembles de ces itemsets sont fréquents, on peut donc considérer que les itemsets fréquents contiennent de nombreuses répétitions. Afin de réduire le nombre des itemsets fréquents découverts, différents algorithmes ont été conçus pour extraire des représentations condensées des itemsets fréquents. Une représentation condensée est un ensemble d'itemsets fréquents réduit qui résume l'ensemble entier des itemsets fréquents. En pratique, la représentation condensée peut être de plusieurs ordres de grandeur plus petite que l'ensemble des itemsets fréquents. En effet, la recherche de ces représentations compressées est souvent plus rapide que la recherche de l'ensemble complet des itemsets fréquents. Parmi les nombreuses représentations condensées proposées pour les itemsets, les plus connues sont : les itemsets fermés, les maximaux et les générateurs. La relation entre ces représentations condensées des itemsets est illustrée dans la figure 2.3.

Les itemsets fermés

Une représentation condensée des itemsets fréquents est constituée des itemsets fermés qui n'ont pas de sur-ensembles ayant le même support. Le problème de l'extraction des itemsets fermés est formellement défini de la manière suivante :

Définition 32

Le problème de la fouille des itemsets fermés, noté par $\mathcal{CIM}(\mathcal{D}, \lambda)$, consiste à énumérer tous les itemsets fermés. C'est-à-dire

$$\mathcal{CIM}(\mathcal{D}, \lambda) = \{I | I \in \mathcal{FIM} \wedge \nexists J \in \mathcal{FIM}\ telque\ I \subset J \wedge Supp(I, \mathcal{D}) = Supp(J, \mathcal{D})\}$$

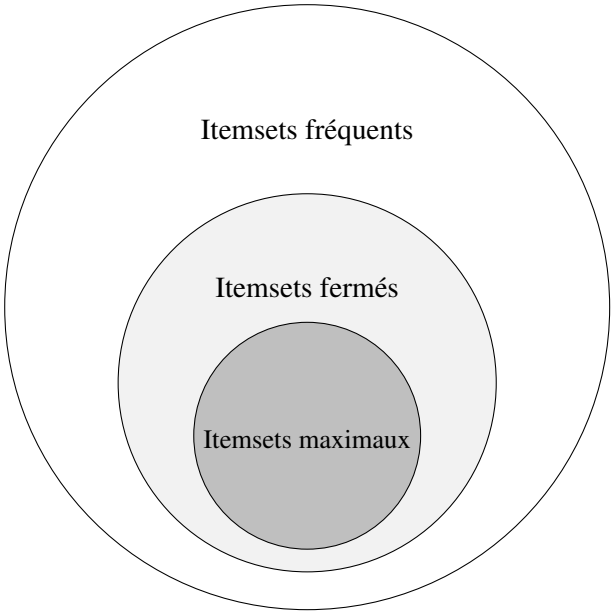


FIGURE 2.3 – Relation entre les représentations condensées des itemsets fréquents : fermés et maximaux.

Exemple

Pour la base de transactions $\mathcal{D}_1$ et $\lambda = 5$, seuls sept parmi les onze itemsets fréquents de la base $\mathcal{D}_1$ sont fermés : $\{b\} : 7, \{f\} : 5, \{g\} : 6, \{b, d\} : 6, \{b, h\} : 5, \{c, g\} : 5$ et $\{e, g\} : 5$.

Propriété 4

Notons que tous les itemsets fréquents ainsi que leurs supports peuvent être déduits à partir de l'ensemble des itemsets fermés, sans les rechercher dans la base de données.

Les approches spécialisées ont étudié les itemsets fermées dans [128, 92, 146, 98, 6, 130], l'algorithme LCM (Linear time Closed itemset Miner) [129, 128, 127] est le plus populaire en raison de son efficacité. D'autre part, des approches déclaratives ont été proposées pour l'extraction des itemsets fermés. Dans [105], une contrainte de fermeture a été proposée pour étendre le modèle CP pour l'extraction des itemsets fermés par l'ajout de la contrainte suivante :

$$\forall a \in \Omega : v_a = 1 \leftrightarrow \sum_{T_i \in \mathcal{D}} t_i(1 - \mathcal{D}_{ia}) = 0 \tag{2.7}$$

La contrainte (2.7) garantit qu'un itemset soit fermé. En effet, si l'item a est présent dans toutes les transactions T_i tel que $t_i = 1$, alors a doit être dans l'itemset i.e., $v_a = 1$.

Pour l'approche SAT, Jabbour et al. [68] ont proposé une extension de leur encodage pour les itemsets fermés en ajoutant la contrainte (2.8).

$$\bigwedge_{a \in \Omega} (p_a \vee \bigvee_{i \in 1..m,\ a \notin T_i} q_i) \tag{2.8}$$

La contrainte (2.8) exprime que si l'ensemble des transactions ne contenant pas l'item a sont fausses, alors l'itemset candidat doit contenir a.

Les itemsets maximaux

Une autre représentation condensée des itemsets fréquents sont les itemsets maximaux. Cette représentation concerne des itemsets fréquents qui n'ont pas de sur-ensembles fréquents.

Définition 33

Le problème de la fouille des itemsets maximaux, noté par $\mathcal{MIM}(\mathcal{D}, \lambda)$, consiste à énumérer tous les itemsets maximaux. C'est-à-dire

$$\mathcal{MIM}(\mathcal{D}, \lambda) = \{I | I \in \mathcal{FIM} \wedge \nexists J \in \mathcal{FIM}\ telque\ I \subset J\}$$

Exemple

Les itemsets maximaux de la base $\mathcal{D}_1$ avec $\lambda = 5$ sont : $\{\{f\} : 5, \{b, h\} : 5, \{c, g\} : 5, \{e, g\} : 5, \{b, d\} : 6\}$.

La représentation des itemsets maximaux est un sous-ensemble de la représentation des itemsets fermés $\mathcal{MIM} \subseteq \mathcal{CIM} \subseteq \mathcal{FIM}$. Par conséquent, le nombre des itemsets maximaux est plus réduit que celui des fermés.

Propriété 5

Les itemsets maximaux peuvent être utilisés pour dériver tous les itemsets fréquents, mais il faut plusieurs balayages de la base de transactions pour récupérer le support de chaque itemset fréquent.

De nombreux algorithmes ont introduit plusieurs techniques pour extraire efficacement l'ensemble des itemsets maximaux [128, 16, 54, 20] en adaptant des méthodes d'extraction des itemsets fréquents (Apriori, FP-growth,...).

Pour capturer les itemsets maximaux, une contrainte de maximalité a été ajoutée dans le modèle CP proposé dans [105] :

$$\forall a \in \Omega : v_a = 1 \leftrightarrow \sum_{T_i \in \mathcal{D}} t_i \mathcal{D}_{ia} \geq \lambda \tag{2.9}$$

Dans [70], la maximalité a été encodée également en SAT en ajoutant la contrainte (2.10) à l'encodage des fermés.

$$\bigwedge_{a \in \Omega} (\neg p_a \rightarrow (\sum_{T_i \in \mathcal{D} | a \in T_i} q_i < \lambda)) \tag{2.10}$$

Les itemsets générateurs

Une représentation condensée des itemsets fréquents sont les itemsets générateurs. Cette représentation concerne des itemsets fréquents qui n'ont pas de sous-ensembles ayant le même support.

Définition 34

Le problème de la fouille des itemsets générateurs, noté par $\mathcal{GIM}(\mathcal{D}, \lambda)$, consiste à énumérer tous les itemsets générateurs. C'est-à-dire

$$\mathcal{GIM}(\mathcal{D}, \lambda) = \{I | I \in \mathcal{FIM} \wedge \nexists J \in \mathcal{FIM}\ telque\ J \subset I \wedge Supp(I, \mathcal{D}) = Supp(J, \mathcal{D})\}$$

La taille de l'ensemble des itemsets générateurs est toujours supérieure ou égale à la taille de l'ensemble des itemsets fermés. Les itemsets générateurs ont été introduits la première fois dans [15]. Récemment, un cadre générique et efficace pour extraire des motifs générateurs en fournissant une recherche en profondeur d'abord a été présenté dans [118].

Dans [19], les auteurs ont proposé la contrainte suivante pour garantir qu'un itemset candidat soit un itemset générateur.

$$\bigwedge_{a \in \Omega} (p_a \rightarrow \bigvee_{T_i \in \mathcal{D} \mid a \notin T_i} (\bigwedge_{b \in T_i} \neg p_b)) \tag{2.11}$$

2.4 Extensions

Dans cette section, nous présentons des travaux conçus pour répondre à des limitations de la tâche d'extraction des itemsets fréquents. Ces améliorations ont conduit à l'apparition de nouvelles variations du problème de la fouille des itemsets. L'introduction de contraintes est une technique pour limiter le nombre des itemsets extraits. Cette technique permet de filtrer les itemsets moins intéressants. Diverses contraintes plus ou moins complexes ont été introduites. Ces contraintes peuvent être appliquées comme une étape de post-traitement après l'extraction de tous les itemsets fréquents. Cependant, cette approche naïve est inefficace. Une meilleure stratégie consiste à inclure des contraintes dans le processus d'extraction pour réduire considérablement la taille de l'espace de recherche et ainsi améliorer les performances. Les premiers algorithmes de la fouille des itemsets qui incluent des contraintes sont basés sur l'algorithme Apriori, qui permettent à l'utilisateur de spécifier des items pouvant être présent ou absent dans les itemsets fréquents, c'est-à-dire les contraintes sont des expressions booléennes sur les items [121].

Une autre limitation de la fouille des itemsets fréquents est qu'elle traite tous les items de la même manière, bien qu'en pratique, certains items soient moins susceptibles d'apparaître dans les itemsets fréquents que d'autres. Cette limitation a conduit à l'étude des items non fréquents dans les bases de transactions ou appelé des items rares [87] et au développement des algorithmes pour résoudre ce problème, en utilisant différents seuils de support qui reflètent les natures des items au lieu d'un seuil de support unique pour tous les items, tels que MSApriori[87], CFPGrowth[65] et CFPGrowth++[76] pour extraire des itemsets fréquents. L'utilisateur peut utiliser ces algorithmes pour sélectionner différents seuils de support pour chaque item. Par conséquent, ces algorithmes peuvent découvrir des itemsets fréquents qui comprennent à la fois des items rares et des items fréquents. De nombreuses recherches ont également été menées pour trouver des itemsets rares dans les bases de données [78, 123, 124] plutôt que des itemsets fréquents, car les ensembles des itemsets fréquents ne sont pas toujours les plus intéressants dans les applications réelles notamment dans les domaines de la médecine, de la finance, et la sécurité. L'extraction de motifs rares est un problème difficile, car les motifs rares sont généralement beaucoup

plus nombreux que les motifs fréquents. Par conséquent, afin d'éviter de découvrir un nombre énorme de motifs rares, diverses définitions de ces motifs ont été proposées, et il existe différentes restrictions sur la définition d'un motif rare. Il existe deux sous-ensembles des itemsets non fréquents : les itemsets rares parfaits [79] et les itemsets rares minimaux [123, 124]. L'algorithme AprioriInverse[79] vise à extraire des itemsets parfaitement rares, dont le support de chaque itemset n'est pas inférieur à un seuil de support minimum et n'est pas supérieur à un seuil de support maximum. En outre, tous les sous-ensembles d'un itemset parfaitement rares doivent également avoir un support qui ne dépasse pas le seuil de support maximal. Mais pour les itemsets rares minimaux qui peuvent être considérés comme des itemsets presque fréquents, il s'agit d'un ensemble des itemsets non fréquents dont tous ses sous-ensembles sont fréquents.

Cependant, certaines recherches ont été menées pour éviter un autre inconvénient des algorithmes spécialisés destinés à être appliqué une seule fois, pour extraire des itemsets à partir des bases de transactions statiques. Dans un environnement dynamique, c'est-à-dire lorsque la base de données de transactions est mise à jour, ces algorithmes doivent être réexécutés pour récupérer les motifs mis à jour. Cette méthode est inefficace car souvent des modifications mineures sont apportées à une base de données, et il n'est pas nécessaire de relancer la recherche de motifs. Pour résoudre ce problème, deux principaux types d'algorithmes ont été développés :

1. *Les algorithmes de fouille incrémentale* sont conçus pour mettre à jour l'ensemble des itemsets fréquents lorsqu'une base de données de transactions est mise à jour par l'insertion de nouvelles transactions, la suppression ou la modification des transactions [77, 82, 85, 134, 122].
2. *Les algorithmes de fouille des flux* ont été proposés pour faire face à un flux potentiellement infini de transactions. Ces algorithmes utilisent des méthodes approximatives pour calculer le support des itemsets et pour traiter les transactions rapidement [21]. estDec [24] et estDec+[116] sont deux méthodes bien connues pour l'extraction des itemsets fréquents dans les flux. Des algorithmes permettant de trouver et de maintenir des itemsets fermés [27] et des itemsets maximaux [36] dans les flux ont également été développés.

Afin de traiter des types de base de données plus complexes, l'extraction des itemsets fréquents a été étendue à l'extraction des itemsets pondérés. Un poids est attribué à chaque item pour indiquer leur importance relative dans la base de données utilisée pour extraire des itemsets pondérés. L'extraction des itemsets pondérés consiste à trouver des itemsets qui ont un poids minimum [142, 139, 140].

L'extraction des itemsets pondérés a été étendue à l'extraction des itemsets à haute utilité, HUIM en abrégé (High-utility itemset mining). La quantité des items est indiquée dans les transactions de la base de données utilisée pour extraire des itemsets à haute utilité. L'objectif de HUIM est de trouver tous les itemsets qui ont une utilité supérieure ou égale à un seuil d'utilité minimum dans une base de données de transactions [89, 84, 86, 88, 42, 148, 143].

2.5 Conclusion

Dans ce chapitre, nous avons discuté le problème de la fouille des itemsets à partir des bases de transactions. Nous avons étudié les principales approches existantes permettant de découvrir des itemsets fréquents, à savoir les approches spécialisées et les approches déclaratives basées sur les contraintes. Nous avons également présenté les différentes variantes de problème autour du problème de l'extraction des itemsets fréquents. En étudiant les approches déclaratives pour l'extraction des itemsets, un défi majeur pour ces approches est relié au passage à l'échelle dû à la taille de l'encodage des grandes bases de transactions. À cet égard, nous proposons de compresser l'encodage SAT pour la fouille des itemsets. Dans le chapitre suivant, nous allons aborder la question de la fouille des motifs séquentiels.

Chapitre 3

Fouille de séquences

Sommaire

Le problème de la fouille de motifs séquentiels fréquents est l'une des tâches importantes en fouille de données. Elle consiste à découvrir les sous-séquences qui apparaissent dans une base de séquences avec une fréquence supérieure ou égale à un seuil spécifié par l'utilisateur. Ce problème peut être vu comme une extension du problème de la fouille d'itemsets.

Dans ce chapitre, nous allons introduire le problème de la fouille de séquences et les différentes techniques existantes pour sa résolution. Dans la section 3.1, nous décrivons brièvement les propriétés des séquences et des bases séquentiels puis nous définissons le problème de la fouille de séquences. Ensuite, la section 3.2 présente un aperçu autour de l'algorithmique utilisée pour résoudre le problème de fouille de séquences. Enfin, dans la section 3.4 nous résumons les différentes extensions de la découverte de motifs séquentiels fréquents.

3.1 Motifs séquentiels

Dans cette section, nous allons présenter les définitions et notations relatives aux séquences suivies par quelques exemples illustratifs.

Nous considérons un ensemble fini et non vide d'items Ω. Nous rappelons qu'un itemset est défini comme un ensemble fini d'items distincts $X \subseteq \Omega$. Le cardinal d'un itemset est noté par $|X|$. Un itemset X est un k-itemset s'il contient k items, c'est-à-dire sa taille est égale à k.

Commençons par la définition d'une séquence.

Définition 35

Une séquence est un tuple ordonné d'itemsets notée $s = \langle a_1, a_2, \ldots, a_n \rangle$ sachant que $a_j \subseteq \Omega$ $(1 \leq j \leq n)$.

Exemple

Par exemple, considérons la séquence $\langle\{a,b,c\},\{d\},\{e,f\},\{g,h,i\}\rangle$ qui représente quatre transactions effectuées par un client. Les items entre accolades représentent un ensemble d'articles (itemset) achetés en même temps. Cette séquence indique qu'un client a acheté les articles a, b et c en même temps, puis il a acheté l'article d, avant d'acheter les articles e et f en même temps, et enfin l'achat g, h et i.

Nous allons maintenant définir une dérivée d'une séquence : une sous-séquence.

Définition 36

Soit s_a une sous-séquence de s_b, c'est-à-dire $s_a \sqsubseteq s_b$, si la séquence s_a est contenue dans la séquence s_b. Une séquence $s_a = \langle a_1, \ldots, a_n\rangle$ est dite contenue dans une autre séquence $s_b = \langle b_1, \ldots, b_m\rangle$ s'il existe des entiers $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ tel que $a_k \subseteq b_{i_k}$ $(1 \leq k \leq n)$.

Exemple

Soit $s_a = \langle\{a\},\{e\},\{g,i\}\rangle$ et $s_b = \langle\{a,b,c\},\{d\},\{e,f\},\{g,h,i\}\rangle$ deux séquences. s_a est une sous-séquence de s_b car $\{a\} \subseteq \{a,b,c\}$, $\{e\} \subseteq \{e,f\}$ et $\{g,i\} \subseteq \{g,h,i\}$. Par contre la séquence $\langle\{a\},\{b\}\rangle$ n'est pas une sous-séquence de $\langle\{a,b\}\rangle$.

Nous rappelons maintenant quelques éléments en lien avec la fouille de séquences.

Définition 37

Une séquence $s = \langle a_1, a_2, \ldots, a_n\rangle$ est dite k-séquence si elle contient k items.

Exemple

Par exemple, la séquence $\langle\{a,b,c\},\{d\},\{e,f\},\{g,h,i\}\rangle$ contient 9 items alors c'est une 9-séquence.

Définition 38

Une base de données de séquences Δ est un ensemble de tuples (sid, s), où sid est un identifiant de séquence et s est une séquence.

Nous utilisons $\mathcal{S}_{id}(\Delta) = \{sid|(sid, s) \in \Delta\}$ pour exprimer l'ensemble des identifiants de séquences associé à la base de données de séquences Δ. Nous notons également les itemsets non singletons dans Δ par $S_{iset}(\Delta) = \bigcup_{(sid,s)\in\Delta}\{a|a \in s, |a| \geq 2\}$.

Considérons la base de données des séquences Δ_1 présentée dans la table 3.1 et définie sur l'ensemble des items $\Omega = \{a, b, c, d, e, f, g\}$.

sid	Séquence
1	$\langle\{a, b\}, \{c\}, \{d, e\}, \{f\}, \{g\}\rangle$
2	$\langle\{b, c\}, \{a\}, \{d\}, \{e, f, g\}\rangle$
3	$\langle\{b\}, \{c\}, \{d, e\}, \{f\}\rangle$
4	$\langle\{b\}, \{d, e\}, \{f\}\rangle$

TABLE 3.1 – Exemple de base de séquences Δ_1.

Le support d'une séquence est défini par :

Définition 39

Le support d'une séquence s_a dans une base de données de séquences Δ est le nombre de séquences dans la base contenant la séquence s_a, c'est-à-dire $support_\Delta(s_a) = |\{(sid, s)|(sid, s) \in \Delta \wedge (s_a \sqsubseteq s)\}|$.

Exemple

Soit $s = \langle\{a\}, \{e\}\rangle$. Le support de s dans la base de séquences présenté dans la table 3.1 est $support_{\Delta_1}(s) = 2$.

Une autre représentation du support d'une séquence s_a communément utilisé dans la littérature est appelé support relatif, qui est égal au pourcentage du nombre de séquences contenant la séquence s_a dans la base de séquences, c'est-à-dire $support_\Delta(s_a) = \frac{|\{(sid,s)|(sid,s)\in\Delta\wedge(s_a\sqsubseteq s)\}|}{|\Delta|}$.

Remarque

Notons qu'une seule occurrence d'une séquence est considérée dans un tuple d'une base de séquences pour le calcul du support. Par exemple la séquence $\langle\{b\}\rangle$ n'est prise en compte qu'une seule fois dans la séquence $\langle\{a\}, \{b\}, \{b, c\}, \{d\}, \{b\}\rangle$.

La fouille de motifs séquentiels, notée par SPM (Sequential Pattern Mining), a pour objectif de découvrir toutes les sous-séquences fréquentes dans une base de données de séquences. Une séquence s_a est appelée un motif séquentiel fréquent ou une séquence fréquente si le support de la séquence s_a satisfait un seuil minimum de support λ, c'est-à-dire si $support_\Delta(s_a) \geq \lambda$. Ce seuil minimum de support λ est un entier positif prédéfini par l'utilisateur. Il permet d'évaluer si un motif est intéressant pour l'utilisateur. Un motif séquentiel fréquent représente la corrélation entre les tuples de la base de données de séquences Δ.

Exemple

Soit $\lambda = 2$, $s = \langle\{a\},\{e\}\rangle$ une séquence fréquente dans la base de séquences de l'exemple 3.1 car $support_{\Delta_1}(s) \geq 2$.

La tâche de la fouille de séquences est considérée comme un problème d'énumération. Son but est d'énumérer tous les motifs avec un support supérieur ou égal au seuil minimum de support fixé par l'utilisateur. Cette énumération de motifs peut être définie comme suit :

Définition 40

Le problème de fouille de motifs séquentiels est le problème d'énumération de l'ensemble de séquences fréquentes, c'est-à-dire $SPM(\Delta, \lambda) = \{s | s \sqsubseteq I \wedge support_{\Delta}(s) \geq \lambda\}$.

Motifs séquentiels	Support
$\langle\{b\}\rangle$	4
$\langle\{c\}\rangle$	3
$\langle\{d\}\rangle$	4
$\langle\{e\}\rangle$	4
$\langle\{f\}\rangle$	4
$\langle\{b\},\{d\}\rangle$	4
$\langle\{b\},\{e\}\rangle$	4
$\langle\{b\},\{f\}\rangle$	4
$\langle\{c\},\{d\}\rangle$	3
$\langle\{c\},\{e\}\rangle$	3
$\langle\{c\},\{f\}\rangle$	3
$\langle\{d,e\}\rangle$	3
$\langle\{d\},\{f\}\rangle$	4
$\langle\{e\},\{f\}\rangle$	3
$\langle\{b\},\{d,e\}\rangle$	3
$\langle\{b\},\{d\},\{f\}\rangle$	4
$\langle\{b\},\{e\},\{f\}\rangle$	3
$\langle\{c\},\{d\},\{f\}\rangle$	3
$\langle\{d,e\},\{f\}\rangle$	3
$\langle\{b\},\{d,e\},\{f\}\rangle$	3

TABLE 3.2 – Motifs séquentiels trouvés dans la base de séquences Δ_1.

La table 3.2 illustre l'ensemble de motifs séquentiels fréquents trouvés dans la base représentée dans la table 3.1. Le seuil de minimum support spécifié λ est de 75%.

Pour explorer l'espace de recherche des motifs séquentiels à l'aide des algorithmes de fouille de séquences, deux manipulations basiques sont effectuées sur les séquences permettant d'étendre une séquence par un item : l'extension de séquence (S-Extension) et l'extension d'itemset (I-Extension). Ces opérations sont utilisées pour générer une $(k+1)$-séquence à partir d'une k-séquence.

Soit la séquence $s_a = \langle a_1, a_2, \ldots, a_n\rangle$ définie comme un préfixe de la séquence $s_b = \langle b_1, \ldots, b_m\rangle$, si $n \leq m$, $a_1 = b_1, a_2 = b_2, \ldots, a_{n-1} = b_{n-1}$ et a_n est égale au premier $|a_n|$ items de la séquence b_n

selon l'ordre lexicographique $\preceq$. L'ajout d'un item à la séquence sous forme d'un nouvel itemset conduit à une extension de la séquence.

Définition 41

Une séquence s_a obtenue avec une S-Extension est une séquence s_b admettant la séquence s_a comme un préfixe et un item x qui apparaît dans un itemset $\{x\}$ après tous les itemsets de la séquence s_a, c'est-à-dire s_b est une S-Extension de la séquence $s_a = \langle a_1, a_2, \ldots, a_m \rangle$, si $s_b = \langle a_1, a_2, \ldots, a_m, \{x\} \rangle$.

Exemple

Soit $s = \langle \{a\} \rangle$ une séquence. $s' = \langle \{a\}, \{b\} \rangle$ est une S-Extension de la séquence s avec l'item b.

Dans l'autre cas, l'insertion d'un item à la fin du dernier itemset dans la séquence à étendre conduit à une extension d'itemset.

Définition 42

Une séquence obtenue avec une I-extension est une séquence s_c, si s_a est un préfixe de s_c avec un item x joint au dernier itemset de la séquence s_a, c'est-à-dire s_c est une I-Extension de la séquence $s_a = \langle a_1, a_2, \ldots, a_m \rangle$ avec un item x, si $s_c = \langle a_1, a_2, \ldots, a_m \cup \{x\} \rangle$

Exemple

Soit $s = \langle \{a\} \rangle$ une séquence. $s' = \langle \{a, b\} \rangle$ est une I-Extension de la séquence s avec l'item b.

Nous avons défini formellement dans cette section le problème de la fouille de séquences et les diverses propriétés et caractéristiques des séquences. La section suivante traite les différentes approches de la fouille de séquences.

3.2 Approches de la fouille de séquences

Dans cette section, nous présentons dans un premier temps un aperçu des principales techniques employées par les algorithmes de recherche de motifs séquentiels. Afin d'extraire efficacement les motifs séquentiels, il est nécessaire de développer des algorithmes qui réduisent efficacement l'espace de recherche en évitant l'exploration entière de l'espace de recherche pour toutes les séquences possibles. Ce qui peut réduire considérablement l'espace de recherche. Récemment, une variété d'algorithmes SPM ont été proposés. Tous ces algorithmes de recherche de motifs séquentiels nécessitent une base de données de séquences Δ et un seuil minimum de support λ en entrée et fournissent une liste de motifs séquentiels fréquents comme sortie. Il est important de noter que pour une base de données de séquences et une certaine

valeur de seuil minimum de support, il n'y a toujours qu'une seule réponse correcte à une tâche d'extraction de motifs séquentiels. Partant de ce fait, si tous les algorithmes d'extraction de motifs séquentiels sont exécutés avec les mêmes paramètres sur la même base de données, ils renverront la même collection de motifs séquentiels. Par cette raison, la différence entre les divers algorithmes ne réside pas dans leur résultat, mais dans la manière dont ils trouvent les motifs séquentiels. Pour rechercher efficacement les motifs séquentiels, plusieurs algorithmes utilisent une variété de stratégies et de formats de données. Par conséquent, certains algorithmes sont plus efficaces que d'autres. En général, nous pouvons distinguer deux grandes familles d'approches utilisées dans les algorithmes du problème de la fouille de séquences. La première concerne les algorithmes basés sur l'approche « générer et élaguer ». La deuxième approche est basée sur l'approche de croissance de motifs « pattern-growth ». Ces approches diffèrent également dans leurs méthodes d'exploration de l'espace de recherche. Certains algorithmes utilisent la recherche en largeur et d'autres effectuent une recherche en profondeur.

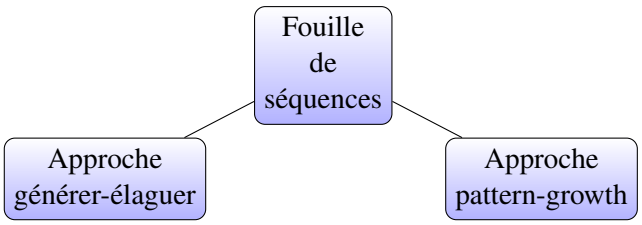


FIGURE 3.1 – Approches de la fouille de séquences

La figure 3.1 illustre les classes des principales approches de la fouille de séquences.

La grande majorité des algorithmes de la fouille de motifs séquentiels appartiennent à la première classe d'algorithmes (c'est-à-dire les méthodes basées sur l'approche générer-élaguer). Ces algorithmes sont basés sur un paradigme de génération et d'élagage de candidats décrit dans la fouille de règles d'association [2] et principalement fondé sur le principe d'Apriori [3], qui affirme que tout sur-motifs d'un motif non-fréquent ne peut pas être fréquent. Ces approches ont le problème de produire de manière répétée un nombre énorme de séquences candidates et de balayer la base de données pour conserver les informations sur le nombre de support pour ces séquences tout au long de chaque itération de l'algorithme, ce qui rend leur coût de calcul très élevé.

Partant de ce fait, une nouvelle approche a été proposé pour résoudre les problèmes de l'approche générer-élaguer. Cette méthode est basée sur le paradigme de croissance de motifs (pattern-growth) et exploite le principe de diviser pour régner pour une extraction efficace des motifs séquentiels. Dans le cas de cette approche, une nouvelle représentation de la base de séquences différente de la représentation verticale ou horizontale est considérée. Dans l'approche pattern-growth, la base de séquences est projetée de manière récursive sur un ensemble de bases de données plus petites basées sur les motifs séquentiels actuels. En explorant uniquement les segments locaux fréquents [58], des motifs sont développés dans chaque base de données de projection. La base de séquences divisée en une collection de bases de données projetées plus petites permet de réduire continuellement l'espace de recherche à parcourir. Le paradigme de pattern-growth élimine la nécessité des étapes de génération et d'élagage des candidats que l'on retrouve dans les algorithmes basés sur Apriori.

L'extraction de motifs séquentiels fréquents offre un large éventail d'applications. Néanmoins, elle présente certaines limites pour des applications particulières. Plusieurs extensions ou variantes du problème de la découverte de motifs séquentiels fréquents ont été proposées pour surmonter ce problème. Ces extensions ont élargi le sujet de l'extraction des séquences fréquentes en utilisant une représentation condensée de motifs telle que les motifs séquentiels fermés et maximaux, des motifs séquentiels multi-niveaux et multi-dimensionnels, des méthodes incrémentales, des méthodes approximatives, l'extraction

de top-k motifs séquentiels.

Nous allons maintenant nous intéresser à la classe des algorithmes qui utilisent l'approche générer et élaguer.

3.2.1 Approche générer et élaguer

Une approche naïve d'extraction de motifs séquentiels consiste à générer toutes les sous-séquences possibles dans la base de séquences. L'ensemble de sous-séquences possibles est très important car une k-séquence contient $2^k - 1$ sous -séquences. Ensuite, l'approche calcule le support de séquences générées et élimine les séquences non-fréquentes. Cependant, une telle approche naïve est inefficace en termes de temps et de mémoire utilisés car le nombre de sous-séquences peut être très important. Pour résoudre ce problème d'explosion de l'espace de recherche, les premières propositions ont utilisé une approche « générer et élaguer » permettant de réduire l'espace de recherche durant l'extraction des motifs. Cette approche est basée sur le paradigme d'Apriori. Un motif non-fréquent est éliminé afin de ne pas générer ses sur-séquences puisqu'ils seront non-fréquents. La première étape de cette approche est la génération des motifs candidats. La seconde étape consiste à calculer le support de l'ensemble des motifs candidats. Plus précisément, l'élagage de l'espace de recherche consiste à éliminer les séquences non-fréquentes de l'ensemble des motifs candidats. Cette approche génère de nouveaux candidats à partir des motifs résultant de l'étape de l'élagage et s'arrête lorsqu'il n'y a aucun motif candidat généré.

Différentes méthodes ont été utilisées pour représenter une base de séquence. Nous focalisons dans un premier temps sur les algorithmes de la méthode de représentation horizontale.

Recherche en largeur avec une représentation horizontale

Une représentation horizontale de la base de séquences est présentée dans la table 3.1. Cette représentation est utilisée pour élaguer l'espace de recherche en largeur.

L'algorithme GSP (Generalized Sequential Patterns)

En 1995, Srikant et Agrawal ont proposé une série d'algorithmes qui ont introduit le problème de motifs séquentiels [1]. Deux algorithmes spéciaux utilisés pour extraire des motifs séquentiels maximaux sont AprioriSome et DynamicSome, et l'algorithme AprioriAll utilisé pour la découverte de tous les motifs séquentiels, qui est une adaptation de l'algorithme Apriori pour les itemsets. En 1996, les mêmes auteurs ont également proposé un algorithme appelé GSP (Generalized Sequential Patterns)[119], qui utilise une base de séquences en format horizontal.

GSP est un algorithme pionnier pour la résolution de la problématique de la fouille de séquences. Cet algorithme est une amélioration des premières propositions d'algorithmes pour l'extraction de motifs séquentiels et qui est basé sur l'algorithme Apriori. L'algorithme GSP génère des séquences candidates en fonction de leurs tailles. Ces candidats sont évalués pour éliminer ceux qui ne sont pas fréquents. Par conséquent, GSP est classé comme un algorithme basé sur la technique générer-élaguer et il procède par une recherche en largeur d'abord. L'algorithme 11 décrit l'algorithme GSP.

Dans un premier balayage de la base de séquences, l'algorithme GSP calcule les 1-séquences fréquentes et les attribue à F_1 (ligne 1), c'est-à-dire qu'à la fin du premier balayage que GSP détermine les items fréquents qui respectent le seuil minimum de support λ en fonction de l'occurrence de chaque item dans la base de séquences. Le processus est itéré en calculant les motifs séquentiels fréquents de taille k, noté (F_k), à partir des séquences de taille $k-1$ noté (F_{k-1}) (lignes 2-8). Au début de k-ième itération, l'algorithme GSP initialise l'ensemble des k-séquences fréquentes F_k par l'ensemble vide (ligne 3). Chaque itération se compose de deux étapes importantes correspondant aux deux fonctions suivantes :

Algorithm 11: GSP Algorithm

Data: Δ : Sequence database, λ : Minimal frequency threshold
Result: The set of all frequent sequences

```
1 F_1 ← {frequent 1-sequence};
2 for (k = 2; F_{k-1} ≠ ∅; k++) do
3 |   F_k ← ∅;
4 |   C_k ← genCandidates(F_{k-1});
5 |   for (c ∈ C_k) do
6 |   |   freq(c) ← compSupports(c, Δ);
7 |   |   if (freq(c) ≥ λ) then
8 |   |   |   F_k ← F_k ∪ {c};
9 return (⋃_k F_k);
```

génération de candidats et calcule de support. Afin de préciser le fonctionnement de GSP en détail, nous décrivons ces deux phases essentielles dans l'algorithme :

— *genCandidates* : les séquences candidates C_k sont générées en joignant les $(k-1)$-séquences fréquentes si possible (ligne 4). L'objectif de cette étape est de générer le moins de candidats possibles tout en maintenant la complétude de l'algorithme. Cette jointure nécessite que chaque séquence fréquente s de F_{k-1} se joint à d'autres séquences s' de F_{k-1} si les $k-2$ derniers items de s sont les mêmes que les $k-2$ premiers items de s'. Par exemple, si l'ensemble de séquences fréquentes F_2 est égal aux quatre séquences suivantes : $\langle\{a\},\{b\}\rangle, \langle\{a\},\{c\}\rangle, \langle\{b\},\{c\}\rangle, \langle\{b,c\}\rangle$. Pour obtenir des 3-séquences fréquentes, chaque 2-séquence fréquente doit se joindre avec les autres 2-séquences qui ont le même dernier item avec ceux qui ont cet item en premier. La séquence $s = \langle\{a\},\{b\}\rangle$ se joint à $s' = \langle\{b\},\{c\}\rangle$ puisque le dernier item de s correspond au premier item de s', dans ce cas la séquence candidate résultante $\langle\{a\},\{b\},\{c\}\rangle$ est une S-Extension. La séquence s peut être étendue également avec la séquence $\langle\{b,c\}\rangle$ pour former la séquence candidate $\langle\{a\},\{b,c\}\rangle$ avec une I-extension. La figure 3.2 illustre un exemple de génération de 3-séquences candidates. Il n'y a plus de séquences correspondantes à joindre dans F_2. Ensuite, l'algorithme itère sur l'ensemble des k-séquences candidates, noté C_k (ligne 5).

(a) S-Extension (b) I-Extension

FIGURE 3.2 – Exemple de génération de 3-séquences candidates

— *compSupport* : durant cette fonction, la base de données est analysée pour trouver les supports de chaque k-séquence candidate c dans (C_k) afin de trouver des k-séquences fréquentes (F_k) (ligne 6). La ligne 7 sert à vérifier par la suite si une k-séquence candidate c est fréquente. Une k-séquence candidate ayant un support supérieur ou égal au support minimum λ est ajoutée à l'ensemble F_k (ligne 8). Par conséquent, une k-séquence ayant un support inférieur au seuil minimum de support est éliminée. L'ensemble des k-séquences fréquentes (F_k) est utilisé dans l'itération suivante afin de générer les (k+1)-séquences candidates.

L'algorithme retourne à la fin l'ensemble extrait des motifs séquentiels fréquents (ligne 9).

Recherche en profondeur avec une représentation verticale

Une base de séquences peut être vue en format vertical converti à partir d'une base de séquences horizontale. Une représentation verticale de la base indique les positions d'itemset où l'item apparaît dans une séquence. La figure 3.3 illustre une représentation verticale de la base de séquences représentée horizontalement dans 3.1. Généralement, cette représentation est utilisée pour élaguer l'espace de recherche en profondeur. Un ensemble de listes de positions « poslists » pour chaque item x forme une représentation verticale de la base de données d'entrée. D'une manière générale, étant donné une k-séquence r, sa poslist L(r) suivra la position du dernier item $r[k]$ dans chaque séquence de la base de données. Le support de la séquence r n'est que le nombre de séquences différentes dans lesquelles r apparaît.

a

SID	$itemsets$
1	1
2	2
3	
4	

b

SID	$itemsets$
1	1
2	1
3	1
4	1

c

SID	$itemsets$
1	2
2	1
3	2
4	

d

SID	$itemsets$
1	3
2	3
3	3
4	2

e

SID	$itemsets$
1	3
2	4
3	3
4	2

f

SID	$itemsets$
1	4
2	4
3	4
4	3

g

SID	$itemsets$
1	5
2	4
3	
4	

FIGURE 3.3 – Représentation verticale de la base de séquences Δ_1

L'algorithme SPAM (Sequential Pattern Mining)

En 2002, Ayres et al. ont proposé l'algorithme SPAM [13]. Cet algorithme est la première approche pour extraire de motifs séquentiels à l'aide d'une stratégie de recherche en profondeur. Par conséquent, comparé à la stratégie de recherche en largeur consistant à produire d'abord tous les motifs de longueur un, puis tous les motifs de longueur deux, et ainsi de suite. Cet algorithme produit des motifs séquentiels de différentes tailles. De plus, SPAM est un algorithme différent des autres approches car il utilise une représentation binaire de la base de séquences. Le pseudo-code de SPAM est présenté dans l'algorithme 12. Il effectue un premier balayage de la base de données pour construire un vecteur de bits pour chaque item. Dans ce vecteur, chaque bit représente une séquence de la base. Si l'item existe dans la séquence, le bit correspondant est mis à 1. L'algorithme SPAM utilise deux processus pour générer des motifs séquentiels candidats (S-Extension et I-Extension). En bref, afin d'étendre une séquence avec un item, l'algorithme applique la fonction booléenne AND entre la représentation binaire de la k-séquence et la représentation binaire de l'item à ajouter dans la séquence pour générer une (k+1)-séquence sous forme de vecteur binaire également. Cette représentation permet d'améliorer efficacement le calcul du support.

L'algorithme SPADE

En 2001, Zaki et al. [144] ont proposé L'algorithme SPADE qui est inspiré de l'algorithme Eclat[145] pour l'extraction des itemsets fréquents. Cet algorithme utilise une représentation verticale de la base de séquences différemment aux premières approches GSP et AprioriAll. Chaque item i a un ensemble de

Algorithm 12: SPAM Algorithm

Data: Δ : Sequence database, λ : Minimal frequency threshold
Result: The set of all frequent sequences

```
1  Scan SDB to create V(SBD) and identify F_1, the list of frequent items;
2  for s ∈ F_1 do
3  |  SEARCH(⟨s⟩, F_1, {e ∈ F_1 | e ≻_lex s}, λ);
4  Function SEARCH (pat, S_n, I_v, λ) :
5  |  Output pattern pat;
6  |  S_temp ← I_temp ← ∅;
7  |  for j ∈ S_n do
8  |  |  if the S-Extension of pat is frequent then
9  |  |  |  S_temp ← S_temp ∪ {i};
10 |  for j ∈ S_temp do
11 |  |  SEARCH (the S-Extension of pat with j, S_temp{e ∈ S_temp | e ≻_lex j}, λ);
12 |  for j ∈ I_n do
13 |  |  if the I-extension of pat is frequent then
14 |  |  |  I_temp ← I_temp ∪ {i};
15 |  for j ∈ I_temp do
16 |  |  SEARCH (the I-extension of pat with j, S_temp {e ∈ I_temp | e ≻_lex j}, λ);
17 End Function
```

tuples noté L(i). Cet ensemble est formé d'une paire $\langle SID, pos(i)\rangle$. pos(s) est un ensemble des positions de l'item i dans la séquence SID. Nous notons l'ensemble de tuples d'un item par $L(i)$. Par conséquent, la liste des ensembles de tuples de chaque item forme une représentation verticale de la base de séquences P. D'une manière générale, une k-séquence s maintient l'ensemble des positions du dernier item k de la séquence s également dans un ensemble de tuples $L(r)$. Cette transformation de la base permet de calculer le support d'une séquence s plus efficacement, qui est égal au nombre de séquences différentes dans lesquelles s apparaît, c'est-à-dire $support_\Delta(s) = |L(s)|$. L'algorithme 13 présente le pseudo-code de SPADE. Un processus est répété pour chaque séquence s_a dans la base de séquences P (ligne 2), l'algorithme affecte ces séquences à l'ensemble de séquences fréquentes (ligne 3). Ensuite, la linge 6 sert à faire une jointure entre chaque séquence s_b, dans l'ensemble P, qui sont dans la même classe d'équivalence, de cette manière SPADE génère comme candidats toutes les extensions possibles du préfixe s_a sans balayer de nouveau la base de séquence. Plus précisément une séquence candidate est obtenue seulement s'il y a une intersection entre les identifiants de séquences de s_a et s_b avec l'ordre entre les identifiants de position $pos(s_a[k])$ et $pos(s_b[k])$ (ligne 7). Par exemple, soit deux k-séquence s_a et s_b avec un préfixe commun de taille (k-1), s_a est étendue avec le k-ième item de la séquence s_b. La ligne 8 vérifie si le support de la nouvelle séquence s_{ab} respecte le seuil minimum de support alors le triplet de cette séquence est affecté à un ensemble P_a. Une fois que l'étape de jointure est finie, si l'ensemble des séquences jointes P_a est non vide, SPADE procède la prochaine itération $k+1$ en utilisant comme base les tuples de séquences résultantes de P_a (ligne 11). L'algorithme s'arrête dans le cas où l'ensemble de P_a est vide, c'est-à-dire s'il n'y a aucune jointure possible.

L'utilisation d'une base de séquences en format vertical offre l'avantage d'employer différentes stratégies d'exploration de l'espace de recherche soit en largeur ou en profondeur.

Algorithm 13: $SPADE$ Algorithm

Initial Call : $F \leftarrow \emptyset, k \leftarrow 0, P \leftarrow \{\langle s, L(s)\rangle | s \in \Omega, support_\Delta(s) \geq \lambda\}$
Data: P : vertical sequence database, λ : Minimal frequency threshold, F : Frequents sequences, k : level
Result: The set of all frequent sequences
for $s_a \in P$ **do**
  $F \leftarrow F \cup \{(s_a, support_\Delta(s_a))\}$;
  $P_a \leftarrow \emptyset$;
  for $s_b \in P$ **do**
    $s_{ab} = s_a + s_b[k]$;
    $L(s_{ab}) = L(s_a) \cap L(s_b)$;
    if $support_\Delta(s_{ab}) \geq \lambda$ **then**
      $P_a \leftarrow P_a \cup \{\langle s_{ab}, L(s_{ab})\rangle\}$;
  if $P_a \neq \emptyset$ **then**
    SPADE(P_a, λ, F, k+1);

3.2.2 Approche pattern growth

Les algorithmes de croissance de motifs sont un autre type d'algorithme essentiel pour la découverte de motifs séquentiels, en plus des approches de recherche générer-élaguer en largeur et des algorithmes verticaux. Ces algorithmes de croissance de motifs sont des algorithmes de recherche en profondeur qui ont été proposés pour remédier à un défaut des algorithmes précédemment cités, qui consiste à générer des motifs candidats qui peuvent ou non exister dans la base de données.

Les algorithmes de croissance de motifs évitent ce problème en scannant récursivement la base de données pour ne trouver que les motifs présents dans la base de données. Cependant, l'analyse de la base de données peut être coûteuse. Des approches basées sur des algorithmes de croissance de motifs ont développé le concept de base de données projetée pour réduire le coût des analyses de bases de données [60, 100, 101].

Recherche en profondeur avec représentation de projection

La projection de la base de données peut être effectuée seulement sur les items fréquents. La table 3.3 correspond à la projection de la base de séquences Δ_1 sur l'item c.

sid	Séquence
1	$\langle\{d, e\}, \{f\}, \{g\}\rangle$
2	$\langle\{b, \_\}, \{a\}, \{d\}, \{e, f, g\}\rangle$
3	$\langle\{d, e\}, \{f\}\rangle$
4	$\langle\{b\}, \{d, e\}, \{f\}\rangle$

TABLE 3.3 – Base de séquences de projection de l'item c.

FreeSpan (Frequent pattern projected Sequential pattern mining)

Dans [59], les auteurs ont proposé l'algorithme FreeSpan en s'inspirant de l'algorithme FPGrowth[58]. Cet algorithme est à l'origine d'autres recherches sur la projection de bases de données afin d'extraire de motifs séquentiels fréquents. L'idée générale de l'algorithme consiste à projeter récursivement la base

de séquences sur plusieurs sous-bases basé sur des items fréquents. Ces bases de projection sont plus petites ce qui rend l'énumération des motifs séquentiels plus pratique et plus facile. De plus, l'utilisation de l'approche de croissance de motifs permet de réduire la génération de sous-séquences candidates. Si un itemset X est non fréquent alors toute séquence dont l'itemset projeté est un sur-ensemble de X qui ne peut pas former un motif séquentiel, d'après une propriété de FreeSpan.

PrefixSpan

L'algorithme PrefixSpan proposé dans [102] est également basé sur le concept de projection de la base de données récursive. Les auteurs ont amélioré leur algorithme FreeSpan en définissant un ordre lexicographique entre les items pour éviter de générer des motifs en double. L'objectif principal est de réduire le nombre de motifs candidats générés. Pour atteindre cet objectif, PrefixSpan fournit une analyse des préfixes communs. La première opération de PrefixSpan consiste à analyser la base de données des séquences originales pour calculer le support des items et de trouver les items fréquents. Sur la base de cette analyse, l'algorithme construit des sous-bases de données qui sont des projections de la base d'origine dérivées à partir des préfixes identifiés. Une base de projection d'un préfixe est l'ensemble de séquences dont le motif préfixe apparaît, et tous les items ou itemsets qui existent avant que le motif préfixe n'apparaisse pour la première fois seront éliminés, c'est-à-dire la base de projection contient seulement les sous-séquences suffixes. Par la suite, PrefixSpan analyse la base projetée et calcule le support de différents items appartenant à cette base pour développer le motif séquentiel en I-Extension ou S-Extension, pour chaque base. L'approche de croissance de motifs employée dans l'algorithme PrefixSpan sert à explorer seulement les motifs existant dans la base contrairement à de nombreux autres algorithmes de fouille de motifs séquentiels. Cependant, PrefixSpan et d'autres algorithmes de croissance de motifs ont l'inconvénient de prendre beaucoup de temps pour analyser de manière répétée la base de données et créer des projections de base de données. En outre, si elle n'est pas mise en œuvre correctement, la création de projections de base de données peut consommer une grande quantité de mémoire, car dans le pire des cas, elle nécessite de copier presque toute la base de données pour chaque projection de base de données. Afin de surmonter ce problème, PrefixSpan emploie une technique d'optimisation « pseudo-projection ». Plus précisément, cette technique consiste à remplacer chaque séquence par un pointeur vers la séquence. Par conséquent, le coût de temps d'exécution et de mémoire est optimisé.

Dans cette section, nous avons présenté deux types principaux des algorithmes de fouille de séquences, des algorithmes basés sur le principe générer-élaguer qui effectuent la génération de candidats en utilisant une représentation horizontale de la base de séquences comme AprioriAll et GSP, des algorithmes de recherche en profondeur qui effectue la génération de candidats en utilisant une représentation verticale de la base de séquences comme SPADE, SPAM. Le deuxième type concerne des algorithmes basés sur le paradigme pattern-growth tel que FreeSpan et PrefixSpan. Dans la section suivante, nous présenterons d'autres variations du problème de la fouille de séquences fréquentes.

3.3 Représentations condensées

Dans cette section, nous allons introduire des représentations condensées proposées pour les motifs séquentiels tels que les motifs séquentiels fermés, maximaux et générateurs.

Motifs séquentiels fermés

Les techniques d'extraction de motifs séquentiels discutée dans la section précédente fonctionnent bien dans les bases de données contenant des séquences fréquentes courtes. Cependant, les performances

de ces algorithmes se dégradent lorsque la fouille de séquences concerne de longues séquences fréquentes ou lorsque des valeurs de seuil minimum de support très faibles sont utilisées. En s'appuyant sur les recherches d'extraction d'itemsets fermés, les chercheurs ont proposé d'étendre les motifs séquentiels fréquents au sujet des motifs séquentiels fermés avec des algorithmes efficaces comme CloSpan [135] et Bide [131].

Une représentation condensée des motifs séquentiels fréquents est les motifs séquentiels fermés qui n'ont pas de sur-ensembles ayant le même support. Le problème de l'extraction des motifs séquentiels fermés est formellement défini de la manière suivante :

Définition 43

Le problème de la fouille des motifs séquentiels fermés, noté par $\mathcal{CSM}(\Delta, \lambda)$, consiste à énumérer tous les motifs séquentiels fermés. C'est-à-dire $\mathcal{CSM}(\Delta, \lambda) = \{s_a | s_a \in \mathcal{SPM} \wedge \nexists s_b \in \mathcal{SPM}\ telque\ s_a \sqsubset s_b \wedge support_\Delta(s_a) = support_\Delta(s_b)\}$

Exemple

Considérons la base de séquences Δ_1 et $\lambda = 75\%$, les motifs séquentiels fermés sont : $\{\langle\{b\},\{e\}\rangle : 4, \langle\{c\},\{e\}\rangle : 3, \langle\{b\},\{d\},\{f\}\rangle : 4, \langle\{c\},\{d\},\{f\}\rangle : 3, \langle\{2\},\{d,e\},\{f\}\rangle : 3\}$.

Le premier algorithme CloSpan (Closed Sequentiel pattern mining)[135] est une extension de l'algorithme PrefixSpan. CloSpan est construit sur le concept de croissance de motifs séquentiels. Il extrait efficacement les motifs séquentiels fermés en fouillant les parties partagées des bases de données projetées pendant le processus d'extraction et de l'élagage de tout espace de recherche redondant, ce qui améliore considérablement l'efficacité de l'extraction. L'idée derrière cette approche est de stocker les séquences générées d'abord dans un arbre de séquences lexicographique en utilisant les mécanismes I-Extention et S-extension et ensuite extraire toutes les séquences fréquentes (fermées et non fermées).

CloSpan introduit une condition d'élagage de l'espace de recherche tout au long de ce processus d'extraction : lorsqu'il identifie deux bases de données projetées à base du même préfixe, il peut cesser d'étendre ce préfixe, il s'agit d'un processus d'élagage des bases de données projetées équivalentes. Les auteurs démontrent que comparer les tailles de deux bases de données de projections permet de déterminer si les deux bases de données de projection sont identiques pour deux séquences dont l'une est une sous-séquence de l'autre.

Dans le deuxième algorithme Bide (BI-Directional Extension)[131], les motifs séquentiels fermés sont extraits sans avoir à maintenir les candidats. Cet algorithme propose un processus d'élagage plus efficace car il vérifie la fermeture d'une séquence à l'aide d'une extension dans deux directions. L'extension vers l'avant (forward extension) est une extension traditionnelle I-Extension en ajoutant l'item à la fin de l'itemset. La seconde direction de l'extension vers arrière (backward extension) est utilisée également pour étendre la séquence, le nouvel item est situé au début de l'itemset ou entre deux autres items. Dans le cas où une séquence ne pourrait pas être étendue vers l'avant ou vers l'arrière donc cette séquence est fermée.

Des algorithmes récents pour l'extraction de motifs séquentiels fermés tels que Clasp [51], CloFast [46] et CM-Clasp [41] utilisent une représentation verticale de la base de séquence. Par conséquent, ces algorithmes sont plus performants que les algorithmes précédents.

Motifs séquentiels maximaux

Les motifs séquentiels sont également étendus vers des motifs séquentiels maximaux. Les motifs séquentiels maximaux sont des motifs séquentiels fréquents qui n'ont pas de sur-ensembles fréquents. Le problème de l'extraction des motifs séquentiels maximaux est formellement défini de la manière suivante :

Définition 44

Le problème de la fouille des motifs séquentiels maximaux, noté par $\mathcal{MSM}(\Delta, \lambda)$, consiste à énumérer tous les motifs séquentiels maximaux. C'est-à-dire $\mathcal{MSM}(\Delta, \lambda) = \{s_a | s_a \in \mathcal{SPM} \wedge \nexists s_b \in \mathcal{SPM}\ telque\ s_a \sqsubset s_b\}$

Exemple

Considérons la base de séquences Δ_1 et $\lambda = 75\%$. Seulement trois motifs séquentiels fréquents sont des motifs séquentiels maximaux : $\{\langle\{c\},\{e\}\rangle : 3,\ \langle\{b\},\{d,e\},\{f\}\rangle : 3,\ \langle\{c\},\{d\},\{f\}\rangle : 3\ \}$.

Propriété 6

L'avantage de cette variation de motifs est qu'ils peuvent être utilisés pour acquérir tous les motifs séquentiels fréquents sans balayer la base de données, mais leur support ne peut être récupéré qu'en effectuant une analyse supplémentaire de la base de données [45].

Pour l'extraction de motifs séquentiels maximaux, plusieurs algorithmes ont été proposés [45, 39, 49, 55, 91], notamment des algorithmes de recherche en largeur comme AprioriAdjust [91]), des algorithmes de croissance de motifs tels que MaxSP [39] et des algorithmes verticaux (par exemple VMSP [45]). En outre, des techniques approximatives telles que DIMASP [49] ont été présentées.

Motifs séquentiels générateurs

Des travaux ont également été menés pour l'extraction des motifs séquentiels générateurs. Ces motifs sont l'ensemble de motifs séquentiels fréquents qui n'ont pas de sous-ensembles ayant le même support. Le problème de l'extraction des motifs séquentiels générateurs est formellement défini de la manière suivante :

Définition 45

Le problème de la fouille des motifs séquentiels générateurs, noté par $\mathcal{GSM}(\Delta, \lambda)$, consiste à énumérer tous les motifs séquentiels générateurs. C'est-à-dire $\mathcal{GSM}(\mathcal{D}, \lambda) = \{s_a | s_a \in \mathcal{SPM} \wedge \nexists s_b \in \mathcal{SPM}\ telque\ s_b \sqsubset s_a \wedge support_\Delta(s_a) = support_\Delta(s_b)\}$

Exemple

Considérons la base de séquences Δ_1 et $\lambda = 75\%$. Les motifs séquentiels générateurs sont : $\{\langle\{c\}\rangle : 3, \langle\{d, e\}\rangle : 3, \langle\{5\}, \{6\}\rangle : 3\ \}$.

De plus, les générateurs peuvent être combinés avec des motifs fermés pour générer des règles avec un antécédent minimum et un conséquent maximum, ce qui permet de dériver la quantité maximale d'informations sur la base à partir d'une quantité minimale d'informations [44]. GenMiner [90], FEAT [48] et FSGP [136] sont des algorithmes de croissance de motifs pour l'extraction de motifs séquentiels générateurs. Un algorithme plus récent nommé VGEN [44], qui étend l'algorithme CM-Spam, s'est révélé plus performant que FEAT et FSGP.

3.4 Extensions

Les algorithmes basés sur la génération de candidats et les algorithmes de croissance de motifs pour l'extraction de motifs séquentiels fréquents ont été couronnés de succès, les chercheurs sont encouragés à étudier comment étendre ces méthodes pour gérer des tâches d'extraction de motifs plus complexes.

Dans cette section, nous allons introduire quelques extensions des méthodes d'extraction de motifs séquentiels fréquents.

L'extraction de motifs séquentiels multidimensionnels est une extension intéressante du problème de l'extraction de motifs séquentiels [38, 103, 117], qui considère une sorte de base de données de séquences plus avancée dans laquelle chaque séquence peut être étiquetée avec une dimension ayant des valeurs symboliques. Elle consiste à énumérer les motifs séquentiels liés à différentes valeurs de dimensions.

Cependant, certains travaux mettent en évidence l'une des limites des algorithmes conçus pour l'extraction des motifs séquentiels à partir des bases de séquences statiques. Ces algorithmes sont destinés à être appliqués une seule fois à une base de données de séquences pour obtenir des motifs. Si la base de données est ensuite mise à jour, les algorithmes doivent être réexécutés pour récupérer les motifs mis à jour. Cette méthode est inefficace car, de temps en temps, seules des modifications mineures sont apportées à une base de données qui ne nécessitent pas de relancer la recherche de motifs. Plusieurs techniques d'extraction de motifs séquentiels incrémentaux ont été développées pour résoudre ce problème[63].

Les algorithmes d'extraction de flux comme IncSPAM [63], SPEED[106], eISeq [23] sont une forme d'algorithme incrémental qui a été conçu pour extraire des motifs séquentiels dans un flux de séquences éventuellement infini [23, 63, 106]. Des algorithmes ont également été conçus pour extraire des motifs séquentiels maximaux comme SPEED [106] et de motifs séquentiels fermés dans des flux de données tels que l'algorithme Seqstream [25].

Une autre extension de l'exploration de motifs séquentiels qui est l'extraction de motifs séquentiels top-k [43]. Il s'agit de découvrir les k motifs séquentiels les plus fréquents dans une base de données de séquences. Ce problème est intéressant car il est souvent difficile pour les utilisateurs n'ayant aucune connaissance préalable de la base de données d'établir le seuil minimum de support en utilisant les algorithmes typiques de fouille de motifs séquentiels. Si le seuil minimum de support est trop faible, trop de motifs peuvent être trouvés, ce qui rend les algorithmes très lents. Dans l'autre cas si le seuil minimum de support est trop élevé, trop peu de motifs peuvent être trouvés. Des techniques sont proposées pour la découverte de motifs séquentiels top-k surmontent ce problème en permettant aux utilisateurs de spécifier directement la quantité de motifs k à découvrir plutôt que de leur permettre de configurer l'option du seuil minimum de support.

Un autre axe de recherche se penche sur la découverte de motifs séquentiels pondérés. Cela permet d'extraire des motifs séquentiels avec des poids (souvent dans l'intervalle [0,1]) attribués à chaque item pour indiquer leur pertinence relative [22, 108, 141]. Le but de la découverte de motifs séquentiels pondérés est de découvrir des motifs séquentiels qui ont un poids minimum. L'extraction de motifs séquentiels pondérés a été étendue vers l'extraction de motifs séquentiels à haute utilité (High-utility sequential pattern mining HUSPM), qui prend en compte non seulement le poids des items mais aussi leur quantité dans les séquences [4, 7, 80, 137]. L'objectif de HUSPM est de trouver tous les motifs séquentiels qui ont une utilité supérieure ou égale à un seuil minimum d'utilité dans une base de données de séquences.

Des algorithmes d'extraction de motifs séquentiels sont optimisés en intégrant des contraintes afin de trouver des motifs plus intéressants et de réduire le nombre de motifs séquentiels trouvés [38, 99, 119, 62, 64].

Il existe deux méthodes possibles pour intégrer des contraintes. La première approche consiste à les utiliser comme une étape de post-traitement sur l'ensemble de tous les motifs séquentiels pour filtrer les motifs inintéressants. Cependant, l'un des inconvénients de cette stratégie est que l'énumération de tous les motifs séquentiels peut prendre beaucoup de temps et une grande quantité de mémoire. La deuxième approche consiste à intégrer les contraintes dans le processus d'extraction afin d'élaguer l'espace de recherche.

3.5 Conclusion

Dans ce chapitre, nous avons décrit le problème de la fouille de séquences à partir de base de séquences. Cela peut être considéré comme une extension de la fouille des itemsets. Ensuite, nous avons étudié les approches principales utilisées dans les algorithmes permettant de trouver l'ensemble des motifs séquentiels fréquents. Enfin, nous avons présenté différentes variantes du problème de la fouille de séquences.

Deuxième partie

Contributions

Chapitre 4

Vers un encodage compact des tâches d'extraction d'itemset basé sur SAT

Sommaire

Le problème de la fouille des itemsets a été modélisé et résolu à l'aide de la programmation par contraintes (CP) et de la satisfiabilité propositionnelle (SAT). Dans ces deux modèles déclaratifs bien connus, le problème est codé sous la forme d'un réseau de contraintes ou d'une formule propositionnelle. Malheureusement, le passage à des données de grande taille reste un vrai challenge pour ces approches dû à la taille de l'encodage. Dans ce chapitre, nous présentons notre contribution pour réduire la taille de l'encodage basé sur SAT. Dans la section 4.1, nous proposons une représentation plus compacte de cet encodage SAT en réécrivant certaines contraintes clés. Dans la section 4.2, nous prouvons que cette reformulation peut être exprimée comme un problème de compression de matrice booléenne. Ensuite, nous proposons une approche gloutonne qui nous permet de réduire considérablement la taille de l'encodage. Dans la section 4.3, nous fournissons des expérimentations de l'approche proposée.

4.1 Encodage Compact pour la fouille des itemsets

Dans cette section, nous proposons une amélioration de l'encodage basé sur SAT des tâches d'extraction des itemsets. Jetons un coup d'œil à la contrainte de fermeture (2.8). Pour de grandes données réelles, souvent $|T| \ll |\Omega|$. C'est-à-dire, le nombre d'items manquants dans chaque transaction est beaucoup plus grand par rapport à ceux qui y apparaissent. Par conséquent, dans le pire des cas, le nombre de clauses associées à la contrainte (2.8) avoisine $|D| \times |\Omega|$. De toute évidence, lorsque le nombre de transactions est important, un tel nombre de clauses est la principale limitation de l'extensibilité des approches SAT pour la fouille des itemsets fréquents. En outre, les clauses de la contrainte de fermeture peuvent être de grande taille, puisque pour pour un item a, nous devons considérer toutes les transactions ne contenant pas a.

Pour faire face à cet important obstacle, nous proposons une amélioration de l'encodage basé sur la

satisfiabilité propositionnelle décrit dans la section (2.2.2). Remarquons d'abord que la formule (2.4)

$$\bigwedge_{i=1}^{m}(\neg q_i \leftrightarrow \bigvee_{a \notin T_i} p_a)$$

peut être formulée de façon équivalente, en éliminant $\leftrightarrow$, comme la conjonction des deux formules suivantes (4.1) et (4.2) :

$$\bigwedge_{a \in \Omega} \bigwedge_{i \in 1...m \mid a \notin T_i} (\neg p_a \vee \neg q_i) \tag{4.1}$$

$$\bigwedge_{T_i \in D} ((\bigvee_{i \in 1...m \mid a \notin T_i} p_a) \vee q_i) \tag{4.2}$$

La contrainte (4.1) relie chaque item à des transactions où il n'apparaît pas. Cette contrainte peut également être reformulée comme un ensemble d'implications :

$$\bigwedge_{a \in \Omega} (p_a \rightarrow \bigwedge_{i \in 1...m \mid a \notin T_i} \neg q_i) \tag{4.3}$$

La contrainte (4.3) exprime que si p_a est affecté à vrai, alors l'ensemble de toutes les variables booléennes associées aux transactions T_i ne contenant pas a sont propagées à faux. Une première approche pour réduire la taille de l'encodage consiste à détecter des sous-termes en $(\bigwedge_{i \in 1...m \mid a \notin T_i} \neg q_i)$, le côté droit des implications en (4.3), et l'introduction de variables auxiliaires pour représenter ces sous-termes. Pour illustrer une telle méthode, reprenons l'exemple 2.1. Le sous-terme $(\neg q_3 \wedge \neg q_6 \wedge \neg q_9)$ apparaît quatre fois en (4.3).

$$\begin{aligned}
&p_a \rightarrow (\neg q_1 \wedge \neg q_2 \wedge \neg q_5 \wedge \neg q_6 \wedge \neg q_8 \wedge \neg q_{10})\\
&p_b \rightarrow (\neg q_1 \wedge \neg q_6 \wedge \neg q_8)\\
&p_c \rightarrow (\neg q_2 \wedge \neg q_3 \wedge \neg q_4 \wedge \neg q_7 \wedge \neg q_9)\\
&p_d \rightarrow (\neg q_1 \wedge \neg q_5 \wedge \neg q_6 \wedge \neg q_8)\\
&p_e \rightarrow (\neg \mathbf{q_3} \wedge \neg q_4 \wedge \neg \mathbf{q_6} \wedge \neg q_7 \wedge \neg \mathbf{q_9})\\
&p_f \rightarrow (\neg \mathbf{q_3} \wedge \neg \mathbf{q_6} \wedge \neg q_7 \wedge \neg q_8 \wedge \neg \mathbf{q_9})\\
&p_g \rightarrow (\neg \mathbf{q_3} \wedge \neg q_4 \wedge \neg \mathbf{q_6} \wedge \neg \mathbf{q_9})\\
&p_h \rightarrow (\neg q_1 \wedge \neg \mathbf{q_3} \wedge \neg q_4 \wedge \neg \mathbf{q_6} \wedge \neg q_8 \wedge \neg \mathbf{q_9})
\end{aligned}$$

Par conséquent, une nouvelle variable r peut être utilisée pour représenter ce sous-terme. Comme le sous-terme $(\neg q_3 \wedge \neg q_6 \wedge \neg q_9)$ est de polarité positive, il suffit d'ajouter la définition $(r \rightarrow (\neg q_3 \wedge \neg q_6 \wedge \neg q_9))$, et de substituer ce sous-terme par r dans chaque implication qui le contient. La formule qui en résulte est équivalente pour encoder l'ensemble de modèles. Ce processus nous permet de gagner 5 clauses au total soit 12 clauses sont remplacées par 7 clauses : 4 clauses (implications en impliquant r) et 3 clauses binaires (associées à la définition de r). Néanmoins, un tel processus nécessite un algorithme de recherche de sous-termes fréquents.

Dans ce travail, nous proposons une nouvelle approche pour identifier efficacement de tels sous-termes. Notre approche est basée sur la réorganisation des transactions et l'introduction de nouvelles variables offrant un moyen original de compacter l'encodage présenté ci-dessus. Dans la suite, nous indiquons par $g(a)$ (respectivement $f(a)$) l'identifiant de la dernière (respectivement première) transaction contenant a dans la base de données.

Définition 46

Pour un item a, nous dénotons par $f(a)$ (respectivement $g(a)$)) la transaction interne (resp. externe) contenant a. C'est-à-dire $f(a) = \min_{1 \leq i \leq m} \{i \mid a \in T_i\}$ et $g(a) = \max_{1 \leq i \leq m} \{i \mid a \in T_i\}$

En utilisant $f(a)$ et $g(a)$, la formule (4.3) peut être exprimée comme la conjonction de (4.4), (4.5) et (4.6).

$$\bigwedge_{a \in \Omega} (p_a \rightarrow \bigwedge_{i \leq f(a) \mid a \notin T_i} \neg q_i) \tag{4.4}$$

$$\bigwedge_{a \in \Omega} (p_a \rightarrow \bigwedge_{f(a) < i < g(a) \mid a \notin T_i} \neg q_i) \tag{4.5}$$

$$\bigwedge_{a \in \Omega} (p_a \rightarrow \bigwedge_{g(a) \leq i \leq m} \neg q_i) \tag{4.6}$$

En fait, en considérant la formule (4.6), à partir de $g(a)$, toutes les transactions avec des identifiants plus grands, leurs variables associées doivent être propagées à faux. Dans la suite, nous montrons comment une telle propagation peut être capturée différemment en reformulant de manière compacte de telles formules en utilisant des variables supplémentaires.

En associant à chaque transaction T_i une nouvelle variable booléenne r_i, la formule (4.6) peut être reformulée comme la conjonction des formules (4.7) et (4.8) :

$$\bigwedge_{a \in \Omega,\ g(a) \leq i \leq m} (p_a \rightarrow r_{g(a)}) \tag{4.7}$$

$$\bigwedge_{i \in 1 \ldots m-1} (r_i \rightarrow \neg q_i) \wedge (r_i \rightarrow r_{i+1}) \tag{4.8}$$

La formule (4.8) forme une chaîne de propagation permettant, lorsque r_i est affecté à vrai, de propager tout q_i à faux de i à m. Une telle chaîne de littéraux propagés peut être exploitée pour différents items a permettant de réduire la taille d'encodage. Dès lors, pour propager les variables associées aux transactions supérieures à $g(a)$ lorsque p_a est affecté à vrai, il suffit d'ajouter la contrainte $p_a \rightarrow r_{g(a)}$.

De manière similaire, pour capturer les propagations à faux des transactions précédant, $f(a)$ lorsque p_a est affecté à vrai, il suffit d'ajouter une chaîne de propagation dans le sens inverse en utilisant de nouvelles variables s_i associées aux transactions de la table. Par conséquent, la formule 4.4 peut être remplacée par les formules 4.9 et 4.10.

$$\bigwedge_{a \in \Omega,\ i \leq f(a)} (p_a \rightarrow s_{f(a)}) \tag{4.9}$$

$$\bigwedge_{i \in 2 \ldots m} (s_i \rightarrow \neg q_i) \wedge (s_i \rightarrow s_{i-1}) \tag{4.10}$$

Maintenant en considérant à la fois $f(a)$ et $g(a)$, la contrainte de couverture peut être réécrite sous la forme suivante :

$$\bigwedge_{i \in 1..m-1} (r_i \rightarrow \neg q_i \wedge r_{i+1}) \tag{4.11}$$

$$\bigwedge_{i \in 2..m} (s_i \rightarrow \neg q_i \wedge s_{i-1}) \tag{4.12}$$

$$\bigwedge_{a \in \Omega} (p_a \rightarrow r_{g(a)} \wedge s_{f(a)} \wedge \bigwedge_{i \in]f(a), g(a)[, a \notin T_i} \neg q_i) \tag{4.13}$$

Comme pour r_i, les variables booléennes s_i sont ajoutées pour garantir la propagation des variables de transaction vers le haut, c'est-à-dire si s_i est vrai, alors toutes les transactions q_j telles que $j \leq i$ sont propagées à faux.

Notons que la réduction de la taille d'encodage dépend de l'ordre des transactions choisi.

Notons également qu'en exploitant le nouveau encodage de la couverture et la contrainte de fermeture (2.8), on peut déduire la contrainte suivante :

$$\bigwedge_{a \in \Omega} (p_a \vee \neg s_{f(a)} \vee \neg r_{g(a)} \vee \bigvee_{i \mid a \notin T_i,\ f(a) < i < g(a)} q_i) \tag{4.14}$$

En effet, la formule (4.14) est dérivée de la formule (2.8) en remplaçant tous les littéraux q_i avec i inférieure à $f(a)$ par $\neg s_{f(a)}$ et celles supérieures à $g(a)$ par $\neg r_{g(a)}$.

Rappelons que notre objetfif est de réduire la taille de l'encodage tout en améliorant le processus de propagation. La taille de l'encodage dépend de l'ordre des transactions. Pour un ordre donné sur les transactions de $\mathcal{D}$, le nombre de clauses dans le pire cas de notre transformation peut être exprimé par la formule suivante :

$$\sum_{a \in \Omega} (g(a) - f(a) + 1) + 3 \times |\Omega| + 4 \times |\mathcal{D}|$$

En effet, chacune des formules (4.11) et (4.12) admet $2 \times |D|$. La formule (4.14) est en forme clausal contenant $|\Omega|$ clauses. Enfin, la formule (4.13) est une implication qui est constituée de $2 \times |\Omega| + \sum_{a \in \Omega} (g(a) - f(a) + 1)$ clauses binaires.

Remarque

Notons que le meilleur ordre possible correspond à ceux où les transactions contenant chaque item sont contiguës. Dans ce cas, la meilleure taille d'encodage est égale à $3 \times |\Omega| + 4 \times |D|$.

Notre problème d'optimisation consiste à minimiser $\sum_{a \in \Omega} (g(a) - f(a))$. La difficulté de trouver le meilleur ordre de transaction peut être vue comme un problème de compression de matrice booléenne. En effet, en considérant une formulation en 0-1 de la base de données des transactions, notre objectif est de réarranger les lignes de la matrice de telle sorte que les 1 valeurs de chaque ligne soient aussi consécutives que possible, c'est-à-dire rendre l'intervalle $]f(a), g(a)[$ aussi réduit que possible. Notre problème est une généralisation du problème bien connu de l'arrangement linéaire optimal [50]. Nous notons ce problème GOLA pour l'arrangement linéaire optimal généralisé.

Définissons formellement notre problème de matrice de compression booléenne.

Définition 47

Soit M une (0-1)matrice. Le problème optimal de la compression de matrice consiste à trouver une permutation σ sur des lignes de M telles que $\sum_{1 \leq i \leq m} h(i)$ est minimisé, où $h(i) = (g(i) - f(i))$. où $f(i)$ (respectivement $g(i)$) représente le premier (respectivement le dernier) 1 qui apparaît dans la colonne i.

Exemple

Reconsidérons la table 2.1. La représentation de la table $\mathcal{D}_1$ en (0-1)matrice est illustrée dans la table 4.1. En réorganisant les transactions de la matrice, la matrice résultante est représentée dans la table 4.2.

	a	b	c	d	e	f	g	h
t_1	0	0	1	0	1	1	1	0
t_2	0	1	0	1	1	1	1	1
t_3	1	1	0	1	0	0	0	0
t_4	1	1	0	1	0	1	0	1
t_5	0	1	1	0	1	1	1	1
t_6	0	0	1	0	0	0	1	0
t_7	1	1	0	1	0	0	0	1
t_8	0	0	1	0	1	0	1	0
t_9	1	1	0	1	0	0	0	0
t_{10}	0	1	1	1	1	1	1	1

TABLE 4.1 – Matrice booléenne de la base de transactions $\mathcal{D}_1$.

Proposition

GOLA est un problème NP-difficile.

	a	b	c	d	e	f	g	h
t_3	1	1	0	1	0	0	0	0
t_9	1	1	0	1	0	0	0	0
t_7	1	1	0	1	0	0	0	1
t_4	1	1	0	1	0	1	0	1
t_2	0	1	0	1	1	1	1	1
t_{10}	0	1	1	1	1	1	1	1
t_5	0	1	1	0	1	1	1	1
t_1	0	0	1	0	1	1	1	0
t_8	0	0	1	0	1	0	1	0
t_6	0	0	1	0	0	0	1	0

TABLE 4.2 – Matrice booléenne de la base de transactions $\mathcal{D}_1$ réordonnée.

Preuve

Notons que Garey et al. [50] ont prouvé que le problème classique de l'arrangement linéaire optimal (OLA en bref) est NP-difficile. La version de décision de ce problème est définie comme suit : étant donné un $G = (V, E)$ et un entier positif k, la question est de répondre s'il existe une bijection $f : V \rightarrow \{1, 2, \ldots, |V|\}$ de sorte que $\sum_{(u,v)\in E} |f(u) - f(v)| \leq k$. Il est facile de remarquer que notre problème généralise l'arrangement linéaire optimal. La solution de GOLA est exactement celle de l'arrangement linéaire optimal. Les arêtes représentent les lignes et les sommets représentent les colonnes.

Plusieurs travaux dans la littérature, ont abordé le problème de la compression d'une matrice booléenne. Par exemple, Booth et Lueker [18] ont montré en 1976 que pour une matrice M donnée, il existe un algorithme en temps linéaire qui détermine si M possède la propriété des un-consécutifs. Cela consiste à prouver s'il est possible de permuter les lignes de sorte que dans chaque colonne, tous les uns soient consécutifs. Ainsi, si la relation possède la propriété des un-consécutifs, nous pouvons réorganiser les colonnes de sorte à pouvoir accéder aux items de chaque ligne en une seule recherche. Cependant, cela ne sera généralement pas possible et il est difficile de minimiser le nombre de recherches lorsqu'une matrice ne possède pas la propriété des un-consécutifs.

4.2 Problème de l'arrangement linéaire optimal généralisé

Dans cette section, nous discutons du problème GOLA en utilisant des approches complètes. Notons qu'en ce qui concerne le problème OLA, il est possible d'envisager certaines heuristiques dédiées permettant un taux de compression élevé en recherchant les meilleures valeurs possibles de $\sum_{a\in\Omega}(g(a) - f(a))$. Pour obtenir la solution optimale, une approche possible consiste à coder le problème vers Max-SAT partielle afin de bénéficier des algorithmes de l'état de l'art, constamment améliorés ces dernières années. Pour ce faire, nous considérons une bijection entre l'ensemble des transactions $\{T_1, \ldots, T_m\}$, les lignes de la matrice, et les positions. Pour chaque transaction T_i, m nouvelles variables booléennes sont introduites pour capturer les positions possibles dans la solution optimale recherchée de T_i, c'est-à-dire t_{ij} est vrai si et seulement si T_i est placé en position j.

La combinaison des contraintes (4.15) et (4.16) exprime une bijection entre l'ensemble des lignes et les nouvelles positions c'est à dire que chaque ligne doit être positionnée exactement dans une position de 1 à m. (4.15) exige que chaque transaction soit placée dans une position et (4.16) indique que deux transactions ne peuvent pas être placées dans la même position.

$$\sum_{j\in 1\ldots m} t_{ij} = 1 \quad \text{for all } i \in [1..m] \tag{4.15}$$

$$\sum_{j\in 1\ldots m} t_{ji} = 1 \quad \text{for all } i \in [1..m] \tag{4.16}$$

Pour capturer les bornes associées à chaque item a, nous lui associons deux ensembles de variables, à savoir $x_{a,1}, \ldots, x_{a,m}$. $x_{a,i}$ indique que la transaction i contient l'item a. La contrainte (4.17) permet de lier $x_{a,i}$ à t_{ij}.

$$\bigwedge_{a\in\Omega}\bigwedge_{j=1}^{m}\Big(\bigvee_{T_i \mid a\in T_i} t_{ij} \rightarrow x_{a,j}\Big) \tag{4.17}$$

$f(a)$ et $g(a)$ peuvent être exprimés comme suit :

$$f(a) = \sum_{i=1}^{m} i \times (x_{a,i} \wedge \bigwedge_{1 \leq j < i} \neg x_{a,j})$$

$$g(a) = \sum_{i=1}^{m} i \times (x_{a,i} \wedge \bigwedge_{i < j \leq m} \neg x_{a,j})$$

Enfin, la fonction objective peut être exprimée comme suit :

$$\text{minimize} \quad \sum_{a \in \Omega} (g(a) - f(a)) \tag{4.18}$$

Une solution de (4.15) $\wedge$ (4.16) $\wedge$ (4.17) soumise à la fonction objective (4.18) correspond à un meilleur réarrangement pour la compression matricielle permettant de compresser efficacement notre encodage. Néanmoins, ce problème est NP difficile. Pour contourner la complexité élevée de ce problème, il est judicieux de concevoir une autre approche offrant un facteur de réduction non optimal mais qui est traitable. Pour ce faire, une approche gloutonne peut être utilisée à la place. Elle est définie comme suit.

Nous sélectionnons récursivement l'item a qui est le moins fréquent dans la base de données courante. Ensuite, toutes les transactions contenant a sont placées au début de $\mathcal{D}$. L'objectif est de minimiser $g(a)$. Cette opération est répétée en considérant l'item le moins fréquent dans les transactions restantes c'est-à-dire, $\mathcal{D} \setminus \{T_i,\ a \in T_i\}$. Remarquons que deux stratégies peuvent être considérées : soit choisir l'item le moins fréquent dans $\mathcal{D} \setminus \{T_i,\ a \in T_i\}$ ou dans toute la base de données $\mathcal{D}$.

Exemple

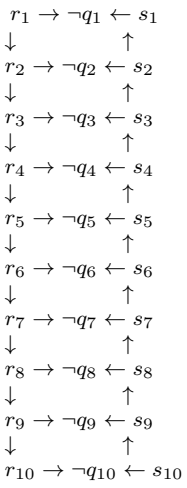
Reconsidérons la base de données de transactions $\mathcal{D}_1$ décrite dans la table 2.1. En appliquant l'approche gloutonne, la base de transactions obtenue est représentée dans la figure 4.1. La chaîne d'implications associée est illustrée sur le côté gauche de la figure.

Enfin, l'encodage complet associé à la base de transactions réordonnées (figure 4.1) est donné par la formule booléenne suivante :

$$\begin{aligned}
&(p_a \rightarrow r_5) \wedge (p_b \rightarrow r_8) \wedge (p_c \rightarrow s_5) \wedge (p_d \rightarrow r_7) \wedge \\
&(p_e \rightarrow \neg q_{10} \wedge s_4) \wedge (p_f \rightarrow r_9 \wedge s_3) \wedge (p_g \rightarrow s_4) \wedge (p_h \rightarrow r_8 \wedge s_2) \wedge \\
&\bigwedge_{5 \leq i \leq 9} (r_i \rightarrow r_{i+1} \wedge \neg q_i) \wedge \bigwedge_{2 \leq i \leq 7} (s_i \rightarrow s_{i-1} \wedge \neg q_i)
\end{aligned}$$

4.3 Évaluation expérimentale

Nous présentons maintenant les expérimentations réalisées pour évaluer les performances de notre approche. En particulier, nous étudions le temps nécessaire pour énumérer les modèles de la formule obtenue par notre approche. Pour effectuer des comparaisons, nous considérons le solveur MiniSAT [35], adapté pour énumérer tous les modèles en effectuant une recherche à base de retours arrière sans redémarrage comme décrit dans l'algorithme 5 de type DPLL. Plus précisément, l'algorithme sélectionne d'abord les variables représentant les items. En effet, chaque affectation sur les variables associées aux



Tid	Itemset
1	a b d
2	a b d
3	a b d h
4	a b d f h
5	b d e f g h
6	b c d e f g h
7	b c e f g h
8	c e f g
9	c e g
10	c g

FIGURE 4.1 – Réorganisation des transactions : une approche gloutonne.

items permet de propager toutes les variables q_i associées aux transactions. Les variables des items sont choisies en fonction de leurs fréquences dans la base de données, c'est-à-dire que le solveur sélectionne en priorité les variables correspondants aux items les moins fréquents. Lors d'un conflit, aucune analyse de conflit n'est effectuée. Un simple retour en arrière est effectué après chaque conflit ou lorsqu'un modèle est trouvé.

Nos expérimentations ont été réalisées sur une machine équipée de processeurs Intel Xeon quad-core avec 32 Go de RAM fonctionnant à 2,66 GHz sous Linux CentOS. Le temps limite a été fixé à 1800 secondes et la mémoire à 10 Go dans toutes les exécutions. Nous considérons les bases de données provenant de FIMI [1] et CP4IM [2]. Les caractéristiques des bases de données considérées sont données dans la table 4.3. Pour chaque base nous fournissons le nombre total de transactions ($\#Transactions$), le nombre d'items distinct ($\#Items$), sa densité (Densité) est égale à $(\frac{|T_i|}{\#Transactions \times \#Items})$ $\times 100$ et la taille du fichier (Size).

Instance	**#Transactions**	**#Items**	**Densité**	**Size**
Chess	3196	75	49.0%	340K
Mushroom	8124	119	19.0%	516K
T10I4D100K	100000	870	1.0%	3.9M
Retail	88162	16470	0.06%	4,2M
Connect	67558	129	35.62%	8.9M
T40I10D100K	100000	942	4.31%	15M
Pumsb	49046	2113	3.0%	16,7M
Kosarak	990002	41267	0.01%	32M
Accidents	340183	468	7.0%	34M

TABLE 4.3 – Caractéristiques des bases de données.

Nous suivons le schéma expérimental de Dlala et al. [31] dont le problème d'énumération des itemsets fréquents est décomposé en énumération d'une séquence de sous-problèmes $\Sigma_i = \Sigma \wedge \neg p_{a_1} \wedge \ldots \wedge$

1. http://fimi.ua.ac.be/data/
2. http://dtai.cs.kuleuven.be/CP4IM/datasets/

$\neg p_{a_{i-1}} \wedge p_{a_i}$ où $\Omega = \{a_1, \ldots, a_n\}$ et Σ est la formule d'encodage basée sur SAT du problème de la fouille des itemsets. Résoudre Σ_i consiste à considérer uniquement les transactions contenant a_i. Ensuite, pour encoder Σ_i, notre approche de compression est utilisée. Dans la suite, nous notons `CESATIM` notre encodage compact pour l'extraction des itemsets SAT en utilisant la décomposition comme dans [31].

Algorithm 14: CESATIM

Input: $\mathcal{D}$: a transaction database, λ : minimum support threshold
Output: S : the set of all Closed Frequent Itemsets $\mathcal{CIM}(\mathcal{D}, \lambda)$

1 $\Omega = \{a_1, \ldots, a_n\} \leftarrow items(\mathcal{D})$;
2 $S \leftarrow \emptyset$;
3 **for** $i \in 1..n$ **do**
4 **if** $Supp(a_i, \mathcal{D}) \geq \lambda$ **then**
5 $\mathcal{D}_{a_i} \leftarrow \{(j, T_j) \in \mathcal{D} \mid a_i \in T_j\}$;
6 $\Gamma \leftarrow \emptyset$;
7 **for** $b \in items(\mathcal{D}_{a_i})$ **do**
8 **if** $Supp(b, \mathcal{D}_{a_i}) < \lambda$ **then**
9 $\Gamma \leftarrow \Gamma \cup \{b\}$;
10 $\Phi \leftarrow encode(\mathcal{D}_{a_i}, \lambda) \wedge p_{a_i} \wedge \bigwedge_{1 \leq j < i} \neg p_{a_j} \wedge \bigwedge_{b \in \Gamma} \neg p_b$;
11 $S \leftarrow S \cup DPLL\_Enum(\Phi)$;
12 **return** S;

L'algorithme 14 résume les principales étapes de notre approche de partitionnement de l'encodage SAT pour l'extraction des itemsets fermés $\mathcal{CIM}$. Il prend en entrée une base de transactions $\mathcal{D}$ définie sur l'ensemble des items Ω, un seuil minimum de support λ, et renvoie en sortie l'ensemble des itemsets fermés. Les items sont triés dans l'ordre croissant en fonction de leurs fréquences (ligne 1). L'ensemble des itemsets fermés $\mathcal{S}$ est initialisé à l'ensemble vide (linge 2). À chaque itération un item a_i est choisi dans Ω (ligne 3). Si son support dans la base $\mathcal{D}$ est supérieur ou égal au seuil minimum de support λ (ligne 4), la sous-base $\mathcal{D}_{a_i}$ de la base $\mathcal{D}$ des transactions qui contiennent l'item a_i est considérée (ligne 5), et un ensemble Γ est initialisé à vide (ligne 6). Ensuite, pour simplifier l'encodage, l'ensemble des items de $\mathcal{D}_{a_i}$ ayant un support inférieur à λ sont stockés dans Γ (ligne 7-9). Par la suite, la formule Φ encodant la table $\mathcal{D}_{a_i}$ est générée conjonctivement avec le littéral positif p_{a_i}, les littéraux négatifs des items non fréquents dans la base $\mathcal{D}$ et les littéraux négatifs des items non fréquents dans la sous-base $\mathcal{D}_{a_i}$. Le solveur $DPLL_{Enum}$ est alors appelé pour trouver tous les modèles de la formule Φ.

Nous comparons les performances de `CESATIM` à `ParaSATMiner` et deux approches CP à savoir tt ClosedPattern [81] et `CoverSize` [112].

Dans la table 4.4, nous présentons les résultats comparatifs de notre approche `CESATIM` en utilisant différentes valeurs de seuil de support minimum. Pour chaque base de données transactionnelles, le nombre de modèles (modèles fermés) et le temps total de CPU (en secondes) sont indiqués. Le symbole (−) est utilisé pour indiquer que le solveur n'a pas été en mesure de terminer le processus d'énumération dans le délai fixé. D'après les résultats, `CESATIM` surpasse `ParaSATMiner` et les approches CP sur presque toutes les instances.

Instance	λ	Closed Pattern	Cover Size	ParaSat Miner	#Clauses	CESATIM	#Clauses	#Models
Retail	80	–	265.10	**12.51**	1617596	13.84	964643	$> 8.10^3$
	60	–	295.47	**16.48**	2091800	18.50	1179210	$> 1.10^4$
	40	–	334.23	**23.53**	3077550	27.45	1591962	$> 2.10^4$
	20	–	439.94	**39.95**	6520011	48.13	2949890	$> 5.10^4$
	10	–	586.16	**72.23**	15415367	88.36	6934878	$> 1.10^5$
Connect	40000	7.54	14.95	6.32	1518140	**3.39**	1065923	$> 7.10^4$
	20000	50.22	75.48	21.51	3205501	**8.45**	2459758	$> 5.10^5$
	10000	526.43	431.19	64.11	5448843	**21.96**	4428425	$> 3.10^6$
	5000	–	–	166.43	7549685	**51.23**	6310490	$> 1.10^7$
Chess	2000	1.51	1.22	0.21	55083	**0.12**	38584	$\simeq 7.10^4$
	1500	6.30	4.09	0.79	128897	**0.38**	96691	$> 5.10^5$
	1000	51.35	28.62	5.04	214078	**2.21**	171067	$> 4.10^6$
	500	577.29	311.47	46.07	339787	**20.37**	285651	$> 45.10^6$
	250	–	–	173.44	414786	**81.92**	352443	$\simeq 2.10^8$
	100	–	–	463.07	465613	**230.13**	395220	$> 5.10^8$
Accidents	100000	101.68	145.96	73.03	14765606	**22.12**	11654191	$\simeq 1.10^5$
	80000	319.25	283.98	142.73	17435639	**35.76**	14024172	$\simeq 4.10^5$
	60000	–	866.21	294.79	22051341	**66.49**	17995214	$> 1.10^6$
	40000	–	–	723.79	31196169	**159.74**	26000938	$\simeq 6.10^6$
Pumbs	40000	20.99	389.43	4.12	428574	**2.30**	288083	$> 2.10^4$
	35000	103.20	325.42	9.76	727145	**4.25**	500954	$\simeq 2.10^5$
	30000	434.25	404.26	29.20	1341404	**10.67**	974194	$\simeq 9.10^5$
	25000	–	994.35	132.86	2111763	**39.65**	1569301	$\simeq 6.10^6$
	20000	–	–	643.11	3801324	**194.70**	2821021	$> 3.10^7$
T40I10D100K	10000	4.16	51.43	2.69	0	**1.48**	0	$\simeq 1.10^2$
	8000	5.08	51.13	4.11	0	**1.88**	0	$> 1.10^2$
	6000	10.38	52.39	7.28	138699	**2.65**	138704	$> 2.10^2$
	4000	30.51	53.26	9.98	1198000	**3.80**	893000	$> 4.10^2$
	2000	144.89	58.22	23.63	16880498	**10.59**	11951802	$> 1.10^3$
T10I4D100K	500	106.87	24.04	3.00	916075	**1.73**	695817	$> 1.10^3$
	400	147.14	25.56	3.60	1709425	**2.21**	1108283	$> 1.10^3$
	300	217.40	27.73	4.61	3345488	**3.02**	1824098	$> 4.10^3$
	200	314.17	29.12	6.51	6226063	**4.39**	3021183	$> 1.10^4$
	100	497.10	32.40	15.24	14354996	**10.79**	6642109	$> 2.10^4$
	50	–	45.05	85.52	34040358	**67.90**	17041570	$> 4.10^4$
Kosarak	4000	–	–	26.81	8298064	**24.22**	5125364	$> 2.10^3$
	3000	–	–	37.35	10867305	**35.41**	6523942	$> 4.10^3$
	2000	–	–	62.27	15819901	**60.15**	9163330	$> 3.10^4$
	1000	–	–	141.90	35068843	**136.87**	20604547	$\simeq 5.10^5$
Mushroom	250	1.14	2.54	1.23	956487	**0.90**	742560	$> 1.10^4$
	100	1.64	1.91	2.16	1149719	**1.65**	868463	$> 3.10^4$
	50	2.70	2.47	2.37	1209497	**1.86**	893076	$> 5.10^4$
	25	2.82	3.16	2.82	1251890	**2.30**	906841	$\simeq 8.10^4$
	5	5.57	4.20	4.31	1300095	**3.74**	917710	$> 1.10^5$

TABLE 4.4 – `ParaSATMiner` vs `CESATIM` vs `ClosedPattern` vs `CoverSize`.

En particulier, notre approche énumère l'ensemble de tous les itemsets fermés de `pumbs` en seulement 194,70 secondes, alors que `ParaSATMiner` a besoin de 643,11 secondes. `ClosedPattern` et `CoverSize` ne sont pas en mesure de terminer l'énumération de tous les motifs dans le temps imparti. Notamment, à l'exception pour la base `Retail` où `ParaSATMiner` surpasse `CESATIM`, sur tous les autres ensembles de données, `CESATIM` obtient les meilleures performances. Pour expliquer les perfor-

mances de notre codage compact, nous reportons dans les colonnes 6 et 8 le nombre total de clauses générées par ParaSATMiner et CESATIM respectivement.

Le nombre de clauses générées par CESATIM correspond au nombre total de clauses de tous les sous-problèmes générés lors des étapes de décomposition. Comme nous pouvons l'observer, en utilisant notre approche, le nombre total de clauses est nettement inférieur. Par exemple, pour les données T10I4D100K, ParaSATMiner génère 34040358 clauses alors que CESATIM n'a besoin que de 17041570 clauses.

4.4 Conclusion

Dans ce chapitre, nous avons présenté une approche efficace pour compacter l'encodage basé sur SAT de la fouille des itemsets fréquents. Nous avons prouvé que cette reformulation compacte peut être exprimée comme un problème de compression de matrice booléenne. Nous avons proposé une approche gloutonne qui a permis de réduire significativement la taille de l'encodage. Les résultats expérimentaux montrent un taux de compression significatif par rapport à la taille originale de l'encodage. Il est intéressant de noter que notre approche permet d'améliorer les performances par rapport à l'approche ParaSATMiner et aux deux approches CP bien connues, à savoir ClosedPattern et CoverSize.

Chapitre 5

Détection et exploitation de symétries dans la fouille de motifs séquentiels

Sommaire

La symétrie présente un grand intérêt dans de nombreux domaines, notamment en informatique, les mathématiques, la physique et la chimie. Les symétries sont des régularités structurelles utiles ou des connaissances qui peuvent être utilisées pour comprendre des entités plus complexes. Dans ce chapitre, nous montrons comment cette relation cachée peut être très utile pour réduire l'espace de recherche dans le cadre de la fouille des motifs séquentiels. Dans la section 5.1, nous commençons par donner une définition de la symétrie dans les bases de séquences. La détection de cette symétrie est présentée dans la section 5.2. Ensuite, nous décrivons comment les symétries peuvent être exploitées pour élaguer l'espace de recherche dans la section 5.3. Nous proposons deux approches pour exploiter ces symétries. La première permet de réduire l'espace de recherche par un élagage dynamique de l'ensemble des motifs candidats possibles dans des algorithmes de type Apriori. La seconde approche élimine les symétries dans une étape de prétraitement en réécrivant la base de séquences. Finalement, dans la section 5.4, nous présentons des expérimentations montrant l'intérêt de supprimer les symétries dans des bases de séquences réelles et synthétiques.

5.1 Symétrie dans la fouille de motifs séquentiels

Dans cette section, nous définissons le concept de symétrie dans l'exploration de motifs séquentiels. Ce concept généralise le concept de symétrie défini dans le cadre du problème de la fouille des itemsets fréquents [72]. Dans la suite, nous adaptons les notations et les définitions proposées dans [72] pour la tâche de fouille de motifs séquentiels. Cependant, il existe une différence principale qui réside dans le fait que dans l'énumération des itemsets, chaque transaction dans la base de données est composée d'un ensemble d'items, alors que pour les motifs séquentiels, chaque transaction est une séquence ou un ensemble ordonné d'itemsets.

Définition 48

Une permutation σ sur un ensemble Ω d'items est une application bijective de Ω à Ω.

Soit $s = \langle a_1, a_2, \ldots, a_n \rangle$ une séquence d'items. Une permutation σ d'une séquence s est définie comme suit : $\sigma(s) = \langle \sigma(a_1), \sigma(a_2), \ldots, \sigma(a_n) \rangle$ où $\sigma(a_j) = \{\sigma(i) | i \in a_j\}$ pour $1 \leq j \leq n$. Puis, pour une base de données de séquences Δ, on définit $\sigma(\Delta) = \{(sid, \sigma(s)) | (sid, s) \in \Delta\}$.

Définition 49

Soit Δ une base de données de séquences. Un renommage f sur $\mathcal{S}_{id}(\Delta)$ est une correspondance bijective entre $\mathcal{S}_{id}(\Delta)$ et $\mathcal{S}_{id}(\Delta)$.

Nous pouvons étendre un renommage f à Δ comme suit : $f(\Delta) = \{(f(sid), s) | (sid, s) \in \Delta\}$.

Définition 50

Soit Δ une base de données de séquences. Une symétrie de Δ est une permutation $\sigma \in \mathcal{P}(\Omega)$ telle qu'il existe un renommage de l'identifiant d'une séquence $\mathcal{S}_{id}(\Delta)$ où $\sigma(\Delta) = f(\Delta)$, c'est-à-dire $f^{-1}(\sigma(\Delta)) = \Delta$.

Chaque symétrie σ peut être exprimée par un ensemble de cycles $\{c_1 \ldots c_n\}$ où chaque cycle $c_i = (a_{i_1}, \ldots, a_{i_k})$ exprime que $\sigma(a_{i_j}) = a_{i_{j+1}}$ pour $j = 1, \ldots, k-1$, et $\sigma(a_{i_k}) = a_{i_1}$. Les cycles de la forme (a, a) sont omis. Nous définissons le nombre de cycles n comme la taille de la symétrie σ, notée $|\sigma|$.

Exemple

Soit la base de données de séquences représentée dans la table 5.1. $\sigma_1 = (a, c)$ et $\sigma_2 = (b, d)(e, f)$ sont deux symétries de Δ_2.

SID	Séquence
1	$\langle \{a, b\}, \{c\}, \{d, e\} \rangle$
2	$\langle \{b, c\}, \{a\}, \{d, e\} \rangle$
3	$\langle \{c, d\}, \{a\}, \{b, f\} \rangle$
4	$\langle \{a, d\}, \{c\}, \{b, f\} \rangle$

TABLE 5.1 – Exemple de base de séquences Δ_2.

$\sigma_1(\Delta_2) = f_1(\Delta_2)$ où f_1 est un renommage de l'identifiant de séquence défini comme suit :

$$f_1(x) = \begin{cases} 2 & \text{si x=1} \\ 1 & \text{si x=2} \\ 4 & \text{si x=3} \\ 3 & \text{si x=4} \end{cases}$$

et $\Sigma_2(\Delta_2) = f_2(\Delta_2)$ où f_2 est un renommage de l'identifiant de la séquence défini comme suit :

$$f_2(x) = \begin{cases} 4 & \text{si x=1} \\ 3 & \text{si x=2} \\ 2 & \text{si x=3} \\ 1 & \text{si x=4} \end{cases}$$

L'intérêt des symétries dans les tâches d'extraction de motifs séquentiels est que les symétries elles-mêmes peuvent être considérées comme une forme de régularité exprimant des connaissances pertinentes dans diverses applications du monde réel.

Supposons que dans la base de séquences Δ_2 (Table 5.1), chaque séquence représente l'ensemble des articles achetés par un client à différents moments. La symétrie $\sigma_1 = (a, c)$ exprime que les articles a et c sont interchangeables dans les différentes séquences, ce qui signifie que les deux articles sont achetés à des moments similaires. Nous pouvons observer des régularités similaires à partir de $\sigma_2 = (b, d)(e, f)$.

En fixant le seuil du support minimum $\lambda = 2$, $s_1 = \langle\{a\}, \{c\}\rangle$ et $s_2 = \langle\{b\}, \{d, e\}\rangle$ sont des motifs séquentiels fréquents de Δ_2. Par symétrie, nous pouvons déduire que $\sigma_1(s_1) = \langle\{c\}, \{a\}\rangle$ et $\sigma_2(s_2) = \langle\{d\}, \{b, f\}\rangle$ sont des motifs séquentiels fréquents de Δ_2. À partir de la définition de la symétrie, nous pouvons déduire la propriété suivante :

Propriété 7

Soit Δ une base de données de séquences et σ une symétrie de Δ. s_a est un motif séquentiel fréquent si et seulement si $\sigma(s_a)$ est un motif séquentiel fréquent.

Supposons que s_a soit fréquent. Cela signifie que $support_\Delta(s_a) = |\{(sid, s)|(sid, s) \in \Delta \wedge (s_a \sqsubseteq s)\}| \geq \lambda$. Comme σ est une symétrie de Δ, nous pouvons en déduire que $support_\Delta(\sigma(s_a)) = |\{(sid, s)|(sid, s) \in \sigma(\Delta) \wedge (\sigma(s_a) \sqsubseteq s)\}| \geq \lambda$. De même, si $\sigma(s_a)$ est fréquent alors en considérant σ^{-1} on déduit que s_a est également fréquent.

Comme la symétrie est une permutation qui laisse la base des séquences invariante, la propriété 7 est valable pour d'autres motifs séquentiels intéressants tels que les motifs fermés et maximaux. Il est intéressant de noter qu'une symétrie induit une relation d'équivalence entre les motifs. Par conséquent, nous pouvons réduire encore l'ensemble des motifs séquentiels, en énumérant un seul motif par classe d'équivalence. Les motifs intéressants restants peuvent être déduits par symétrie. Nous pouvons nous référer à une telle représentation concise réduite en tant que motifs séquentiels fréquents, fermés et maximaux modulo une symétrie.

5.2 Détection de la symétrie dans les bases de séquences

Dans cette section, nous décrivons notre approche de détection de symétries à partir d'une base de données de séquences. L'un des moyens les plus populaires pour découvrir des symétries consiste à réduire le problème à une tâche d'énumération des automorphismes de graphes [8].

Pour prendre en compte l'ordre des items dans les séquences, nous convertissons la base de données de séquences en un graphe orienté coloré, où les sommets sont de trois types (ou couleurs), correspondant respectivement aux items, itemsets et identifiants de séquences. Les couleurs des sommets sont prises en compte lors du processus d'énumération des automorphismes du graphe (c'est-à-dire que les sommets de couleurs différentes ne peuvent pas être permutés entre eux). Par exemple, les sommets correspondant aux items ne peuvent pas être permutés avec les sommets correspondant aux séquences-id. Les couleurs

sont illustrées dans la figure 5.1 par différents types de sommets. Les arêtes relient les items aux itemsets et les itemsets aux identifiants de séquences, tandis que les arcs sont introduits pour exprimer l'ordre entre les itemsets d'une séquence donnée, en reliant séquentiellement les itemsets de la séquence.

Par conséquent, nous réduisons formellement le problème de la découverte de symétries au problème du calcul des automorphismes d'un graphe dirigé coloré.

Définition 51

Un *graphe dirigé coloré* est un triplet $\mathcal{G} = (V, A, \eta)$ avec V un ensemble de sommets et $A \subseteq V^2$ un ensemble d'arcs et η est une fonction de V à N qui associe un entier positif (une couleur) à chaque sommet.

Définition 52

Un automorphisme d'un graphe $\mathcal{G} = (V, A, \eta)$ est une permutation σ de l'ensemble des sommets V, telle que :

1. $\forall v \in V, \eta(v) = \eta(\sigma(v))$;
2. $(u, v) \in A$ si et seulement si $(\sigma(u), \sigma(v)) \in A$.

Autrement dit, un automorphisme d'un graphe dirigé coloré est un isomorphisme du graphe dirigé coloré sur lui-même. L'ensemble de tous les automorphismes de $\mathcal{G}$ est appelé groupe d'automorphisme de $\mathcal{G}$ et est noté $\mathcal{A}(\mathcal{G})$.

Nous montrons maintenant comment une base de données de séquences peut être codée sous la forme d'un graphe dirigé coloré.

Définition 53

Soit Ω un ensemble d'items et Δ une base de séquences sur Ω. Nous définissons le graphe orienté coloré $\mathcal{G}$ associé à Δ par $\mathcal{G}(\Delta) = (V, A, \eta)$ où,

— $V = \Omega \cup \mathcal{S}_{id}(\Delta) \cup \mathcal{S}_{iset}(\Delta)$;

— $A = \{(a, i) | a \in \mathcal{S}_{iset}(\Delta), i \in a\} \cup \{(a, sid) | (sid, s) \in \Delta, a \in s\} \cup_{s=\langle a_1,\ldots,a_n\rangle | (sid,s)\in\Delta} \{\langle a_i, a_{i+1}\rangle | 1 \leq i < n\}$ et,

— η est défini comme suit :

$$\forall v \in V, \eta(v) = \begin{cases} 0 & \text{if } v \in \mathcal{S}_{id}(\mathcal{D}) \\ 1 & \text{if } v \in \mathcal{S}_{iset}(\mathcal{D}) \\ 2 & \text{if } v \in \Omega \end{cases}$$

Pour des raisons de clarté, dans la définition 53, nous avons utilisé à la fois les arêtes et les arcs, désignés par $(.)$ et $\langle . \rangle$ respectivement. Par ailleurs, une arête entre deux sommets peut être représentée par deux arcs. La conversion d'une base de données de séquences Δ_2 (voir table 5.1) en un graphe orienté coloré $\mathcal{G}(\Delta_2)$ est représentée dans la figure 5.1.

Nous distinguons trois types de sommets colorés : les identifiants de séquences représentés par des rectangles de couleurs gris clair, les items représentés par des cercles de couleurs gris foncé et les itemsets

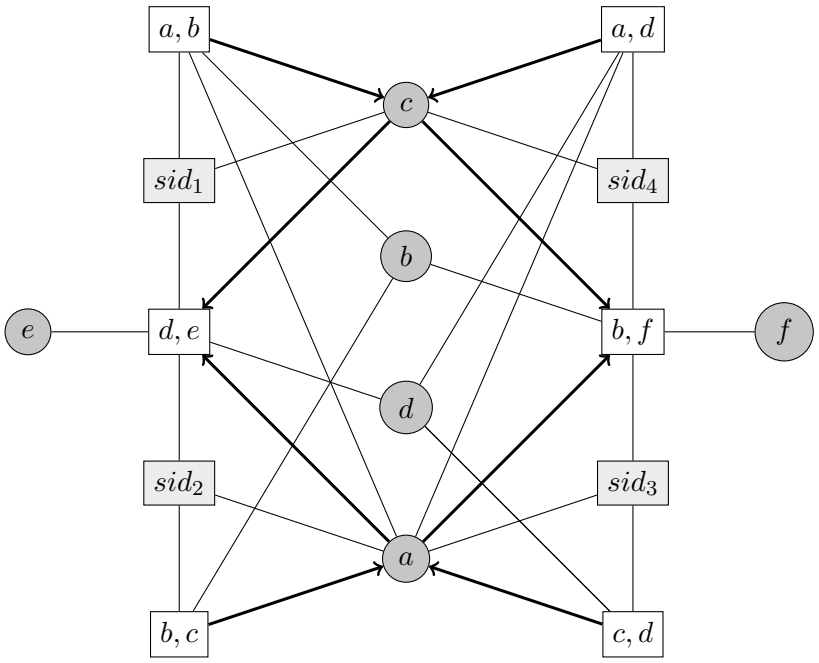


FIGURE 5.1 – De la base de séquences Δ_2 au graphe dirigé coloré $\mathcal{G}(\Delta_2)$

représentés par des rectangles. Illustrons comment la séquence $(sid_1, \langle\{a, b\}, \{c\}, \{d, e\}\rangle)$ est encodée dans le graphe. Tout d'abord, un sommet est associé à la séquence sid_1, relié par des arêtes aux sommets associés aux itemsets $\{a, b\}$, $\{d, e\}$ et $\{c\}$ (représenté par un seul item c). Chaque sommet représentant un itemset est relié par des arêtes aux items qu'il contient. Par exemple, le sommet associé à $\{d, e\}$ est connecté aux sommets associés aux items d et e. Enfin, l'ordre entre les items de la séquence est représenté par des arcs entre les sommets associés aux items $\{a, b\}$ et $\{c\}$, et entre les sommets associés aux items $\{c\}$ et $\{d, e\}$.

Par construction, cette transformation garantit qu'il existe une correspondance entre les symétries de la base de données des séquences et les automorphismes du graphe dirigé coloré comme défini ci-dessous.

5.3 Élagage de symétries

Dans cette section, nous décrivons comment les symétries peuvent être utilisées pour réduire l'espace de recherche des tâches de la fouille des motifs séquentiels. À cette fin, nous proposons deux approches complémentaires. La première vise à exploiter les symétries dans des algorithmes d'énumération de motifs séquentiels de type Apriori, tandis que la seconde est indépendante de l'algorithme, car elle élimine les symétries dans une étape de prétraitement, en réécrivant la base de données séquentielle des transactions.

5.3.1 Exploitation des symétries dans un algorithme de type apriori

Décrivons la première approche. Plus précisément, nous montrons comment les symétries peuvent être intégrées dans l'un des algorithmes de la fouille des motifs séquentiels bien connus, GSP (Generalized Sequential Pattern) basé sur un algorithme de type Apriori (niveau par niveau) [120], décrit dans l'algorithme 11.

Décrivons maintenant comment les symétries peuvent être intégrées dans l'algorithme GSP (voir algorithme 15, abrégé GSP_{Sym}).

Algorithm 15: GSP_{Sym} Algorithm

Data: Δ : Sequence database, λ : Minimal frequency threshold, $\mathcal{S}$: Set of symmetries in Δ
Result: The set of all frequent sequences

$F_1 \leftarrow$ { frequent 1-sequences};
for $(k = 2;\ F_{k-1} \neq \emptyset;\ k++)$ **do**
  $F_k \leftarrow \emptyset$;
  $C_k \leftarrow genNonSymCandidates(F_{k-1}, \mathcal{S})$;
  for $(c \in C_k)$ **do**
    $freq(c) \leftarrow compSupports(c, \Delta)$;
    if $(freq(c) \geq \lambda)$ **then**
      $F \leftarrow genSymFrequentPattern(c, \mathcal{S})$;
      $F_k \leftarrow F_k \cup \{c\} \cup F$;
return $(\bigcup_k F_k)$;

Introduisons quelques notations supplémentaires. Étant donné $\sigma = (x_1, y_1) \cdots (x_k, y_k)$ une symétrie de Δ, nous définissons $in(\sigma)$ comme l'ensemble $\{x_1, \ldots, x_k\}$ et $out(\sigma)$ comme l'ensemble $\{y_1, \ldots, y_k\}$. Étant donné un ordre total $\preceq$ sur les itemsets I, σ est une symétrie ordonnée si $x_i \preceq y_i$ et $x_i \preceq x_j$ pour tout $1 \leq i < j \leq k$. Nous supposons dans la suite que les symétries sont ordonnées lexicographiquement. Sans perte de généralité, nous supposons également que les items des itemsets d'une séquence sont également ordonnés lexicographiquement. Nous pouvons maintenant définir $first^{\sigma}(s)$ comme le premier item de s apparaissant dans σ.

Exemple

Soit $s = \langle\{c\}, \{d, e\}, \{a, b\}\rangle)$ est une séquence, $\sigma_1 = (a, c)$ et $\sigma_2 = (b, d)(e, f)$ sont deux symétries Δ_2. Nous avons $first^{\sigma_1}(s) = c$ et $first^{\sigma_2}(s) = d$.

La propriété suivante nous permet d'exploiter les symétries dans l'algorithme GSP :

Propriété 8

Soit $c \in F_k$ un motif séquentiel candidat et σ une symétrie de Δ. Si $first^{\sigma}(c) \in out(\sigma)$ alors $\exists\, c' \in F_k$ avec $c' \neq c$ un motif séquentiel candidat tel que $\sigma(c') = c$.

Soit $first^{\sigma}(c) = a \in out(\sigma)$. Alors σ peut être réécrit comme $\sigma = (x, a)\sigma'$. Comme σ partitionne l'ensemble des motifs candidats F_k en classes d'équivalence. Cela signifie qu'il existe un motif candidat $c' \in F_k$ tel que $first^{\sigma}(c') = x$ avec $\sigma(c') = c$.

Exemple

Considérons à nouveau la séquence $c = \langle\{c\},\{d,e\},\{a,b\}\rangle$ et la symétrie $\sigma_2 = (b,d)(e,f)$. On a $first^{\sigma_2}(c) = d \in out(\sigma_2)$. En effet, il existe $c' = \langle\{c\},\{b,f\},\{a,d\}\rangle$ tel que $\sigma_2(c') = c$. Par conséquent, c' est fréquent si et seulement si c est fréquent.

La propriété 8 nous permet de réduire l'ensemble des motifs séquentiels candidats de F_k à un ensemble de motifs non symétriques. Plus précisément, nous évitons de vérifier le support de ces motifs séquentiels candidats symétriques sur la base de séquences, qui est l'étape la plus coûteuse dans les algorithmes de type GSP. Cette réduction est mise en œuvre dans la fonction $genNonSymCandidates(F_{k-1}, \mathcal{S})$ (ligne 4). Dans cette fonction, lorsqu'un motif séquentiel candidat c est généré en joignant deux motifs séquentiels fréquents de F_{k-1}, nous vérifions s'il existe $\sigma \in \mathcal{S}$, tel que $first^{\sigma}(c) \in out(\sigma)$. Dans le cas positif, un tel motif candidat est éliminé car il peut être déduit par symétrie. Dans la fonction $genSymFrequentPattern(c, \mathcal{S})$, nous déduisons les autres motifs séquentiels fréquents par symétrie (ligne 8) et les ajoutons à F_k (ligne 9). Comme nous pouvons l'observer, les motifs séquentiels fréquents dans F (ligne 8) sont dérivés par symétrie sans calculer leurs supports dans la base de séquences. De plus, lorsque le motif séquentiel candidat c n'est pas fréquent ($freq(c) < \lambda$) dans (ligne 7), le support de son motif séquentiel symétrique n'est pas calculé.

Remarque

Notons que dans GSP_{Sym}, nous n'exploitons les symétries qu'individuellement. D'autres réductions peuvent être obtenues par composition de plusieurs symétries. En effet, comme l'ensemble des symétries avec un opérateur de composition forme un groupe, composer deux symétries est aussi une symétrie. Cependant, cette question induit un sur-coût important en termes de complexité de calcul.

5.3.2 Élagage de symétries basé sur le pré-traitement

Décrivons maintenant la deuxième approche d'élagage des symétries. Elle vise à éliminer les symétries dans une étape de pré-traitement. Notre approche d'élagage de symétries basé sur le pré-traitement est une extension de l'approche proposée dans [72] pour le problème de la fouille d'itemsets.

Nous présentons ce pré-traitement basé sur les symétries pour la fouille des itemsets [72]. Soit $\sigma_1 = (a,b)$ est symétrie d'une base de transactions $\mathcal{D}$. Les itemsets fréquents possibles impliquant les items de σ_1 peuvent se distinguer en trois cas : itemsets contenant l'item "a" et "b" noté $I_{(a,b)}$, itemsets contenant uniquement l'item "a" noté I_a et les itemsets contenant uniquement l'item "b" noté I_b. Le pré-traitement de symétries proposé dans [72] consiste à supprimer l'item $b \in T_i$ si $a \notin T_i$, c'est-à-dire si l'item a n'est pas présent avec l'item b. Dans un cas général, une symétrie $\sigma = (a_1,x_1),\ldots,(a_n,x_n)$ peut être éliminé en supprimant l'item $\forall k \in 1..n \;\; x_k \in T_i$ si et seulement si les items $\forall j,k \in 1..n \;\; j \leq k, \nexists\, a_j \in T_i$.

La table 5.2 présente le pré-traitement de la symétrie $\sigma = (3,5)(4,6)(7,8)$ sur la base de transations $\mathcal{D}$.

Pour la fouille des séquences, nous définissons $S = \{\sigma_1,\ldots,\sigma_m\}$ l'ensemble des symétries de la base séquentielle Δ, et une symétrie $\sigma \in S$ définie comme suit : $\sigma = (a_1,x_1),\ldots,(a_n,x_n)$. Nous définissons l'ensemble d'entrée (resp. de sortie) des items comme $in(\sigma) = \{a_1,\ldots,a_n\}$ (resp. $out(\sigma) = \{x_1,\ldots,x_n\}$). On note également $in^j(\sigma) = \{a_1,..,a_j\}$ les j premiers items de $in(\sigma)$. Nous associons à chaque item $x_i \in out(\sigma)$ une liste d'items $L^{\sigma}(x_i) = \{a_1,\ldots,a_i\}$. Pour éliminer les symétries de Δ

Tid	Transaction
1	{1,2,3,4}
2	{1,2,5,6}
3	{7,8}
4	{9}

(a) Un exemple de base de transactions $\mathcal{D}$

Tid	Transaction
1	{1,2,3,4}
2	{1,2}
3	{7,8}
4	{9}

(b) Un exemple de base de transactions $\mathcal{D}_{sansSym}$

TABLE 5.2 – Exemple de pré-traitement de symétries sur une base de transactions.

dans une étape de pré-traitement, pour chaque symétrie $\sigma \in S$, et pour chaque séquence de Δ, un item $x_i \in out(\sigma)$ de la séquence est supprimé si et seulement si cet item n'est précédé d'aucun item de sa liste associée $L^{\sigma}(x_i)$. Nous illustrons ce pré-traitement dans l'exemple suivant :

Exemple

Soit $\sigma = (a, c)$ une symétrie de Δ et X une sous-séquence d'items. Les motifs séquentiels possibles impliquant les items de $s_1 = \langle\{a\}, X\rangle$, $s_2 = \langle\{a\}, X, \{c\}\rangle$, $s_3 = \langle\{c\}, X\rangle$ et $s_4 = \langle\{c\}, X, \{a\}\rangle$. Le pré-traitement de Δ en utilisant la symétrie $\sigma = (a, c)$ élimine l'item c des deux séquences s_3 et s_4 car il n'est pas précédé de l'item a. En effet, en appliquant σ sur s_1 et s_2, nous dérivons s_3 et s_4. En résumé, le pré-traitement nous permet d'éviter le calcul de support des deux séquences s_3 et s_4. Ces deux derniers motifs séquentiels peuvent être obtenues par symétrie. L'un des principaux avantages de ce deuxième élagage par symétrie est qu'il est indépendant de l'algorithme d'extraction de motifs séquentiels utilisé.

Soit $\sigma_1 = (a, c)$ et $\sigma_2 = (b, d)(e, f)$ deux symétries de la base de séquences Δ_2. La table 5.3 présente le pré-traitement de σ_1 et σ_2 sur la base de séquences Δ_2.

SID	Séquence
1	$\langle\{a, b\}, \{c\}, \{d, e\}\rangle$
2	$\langle\{b\}, \{a\}, \{d, e\}\rangle$
3	$\langle\{a\}, \{b, f\}\rangle$
4	$\langle\{a\}, \{c\}, \{b, f\}\rangle$

TABLE 5.3 – Exemple de pré-traitement de symétries sur la base de séquences Δ_2.

5.4 Évaluation expérimentale

Notre évaluation expérimentale est menée sur des bases de séquences réelles et synthétiques largement utilisées dans la fouille de motifs séquentiels. Toutes les bases de séquences proviennent de la page web de Philippe Fournier Viger SPMF (URL : http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php). Nous avons également considéré de nouvelles bases de séquences, appelées bases de séquences synthétiques symétriques, dérivées d'une symétrie sur des bases de séquences synthétiques.

— *Les bases de séquences réelles* impliquent des séquences d'items, leurs caractéristiques sont décrites dans la table 5.4. Pour chaque base de séquences réelles, nous fournissons le nombre de séquences ($|\Delta|$), le nombre d'items ($|\Omega|$) et la densité.

Dataset	$\|\Delta\|$	$\|\Omega\|$	densité
ProofSequences	35	13	2.69
Sign	730	267	2.73
Leviathan	5834	9025	0.64
PubMed	17237	19931	0.86
FIFA	20450	2990	6.84
BIBLE	36369	13905	2.61
BMSWebView1	59601	497	119.92
Kosarak	69999	21144	3.31
BMSWebView2	77512	3340	23.20
MSNBC	989818	17	58224.58

TABLE 5.4 – Bases de séquences réelles.

— *Les bases de séquences synthétiques* correspondent à des bases de séquences d'items. Elles sont générées à l'aide du Quest Dataset Generator d'IBM, et converties au format SPMF. Nous considérons les bases de séquences présentes sur la page web SPMF. Les cinq premières bases de séquences, synDataset_1 à synDataset_5, correspondent à des bases de séquences synthétiques, nommées $data.slen\_10.tlen\_1.seq.patlen\_X.lit.patlen\_8.nitems\_5000$ avec $X = 2$ à 6. Les quatre dernières bases de séquences (synDataset_6 à synDataset_9) correspondent aux bases de séquences synthétiques nommées $data.slen\_8.tlen\_1.seq.patlen\_X.lit.patlen\_8.nitems\_5000$ avec $X = 2$ à 5. Les caractéristiques des neuf bases de séquences synthétiques sont décrites dans la table 5.5. Pour chaque base de séquences synthétiques, nous indiquons le nombre de séquences dans la base de données ($|\Delta|$), le nombre d'items ($|\Omega|$), la taille des itemsets ($|iSets|$) et le nombre d'itemsets dans une séquence ($\#iSets$).

Synthetic Datasets	$\|\Delta\|$	$\|\Omega\|$	$\|iSets\|$	#iSets
synDataset_1	50351	41911	30	8
synDataset_2	47784	53137	28	8
synDataset_3	47555	62296	27	9
synDataset_4	47987	69686	30	9
synDataset_5	48466	75476	26	10
synDataset_6	45534	41270	25	6
synDataset_7	45451	52551	28	7
synDataset_8	46130	61137	27	9
synDataset_9	47133	68240	26	10

TABLE 5.5 – Bases de séquences synthétiques.

— *Les bases de séquences synthétiques symétriques* sont générées par symétrie à partir des bases de séquences synthétiques mentionnées ci-dessus. Étant donné une base de séquences synthétique (Δ), nous générons d'abord aléatoirement sans remplacement une permutation σ sur les items de Δ. Soit $\Delta^s = \{(sid_{|\Delta|+i}, \sigma(s_i)) | (sid_i, s_i) \in \Delta\}$. Une nouvelle base de séquence synthétique symétrique $\Delta_{Sym} = \Delta \cup \Delta^s$ est dérivée. De toute évidence, la permutation σ est une symétrie de Δ_{Sym}. Étant donné une base de séquences synthétique synDataset_X, et n la taille de la symétrie

utilisée dans le processus de génération, nous notons la base de séquence synthétique symétrique associée par symSynDataset^n_X.

Pour la méthode de détection de symétrie, nous avons utilisé *bliss* (URL : http://www.tcs.hut.fi/Software/bliss/index.html), qui est un outil libre pour le calcul des groupes d'automorphisme et des formes canoniques des graphes [74].

Nous fournissons plusieurs expérimentations pour d'abord mettre en évidence la présence de symétries dans des bases séquentielles et ensuite pour montrer l'effet de leurs exploitations dans l'élagage de l'espace de recherche.

Notre première expérimentation vise à montrer la présence de symétries dans les bases de séquences d'extraction de motifs séquentiels. Les bases de séquences réelles (voir Table 5.6) et les bases de séquences synthétiques (voir Table 5.7) présentent plusieurs symétries ($\#Sym$) détectées en un temps raisonnable ($time$ en secondes).

Dataset	#sym	time(s)
ProofSequences	0	0.85
Sign	4	5.97
Leviathan	46	0.57
PubMed	72	3.10
FIFA	7	2.35
BIBLE	240	6.57
BMSWebView1	0	3.6
Kosarak	1244	3.08
BMSWebView2	0	4.8
MSNBC	109	1.43

TABLE 5.6 – Détection de symétries sur des bases de séquences réelles.

Synthetic Datasets	#sym	time(s)
synDataset_1	3029	1.77
synDataset_2	3892	2.50
synDataset_3	2096	3.52
synDataset_4	1578	4.52
synDataset_5	1235	5.60
synDataset_6	2748	1.25
synDataset_7	2628	1.96
synDataset_8	1891	2.95
synDataset_9	1836	3.62

TABLE 5.7 – Détection de symétries sur des bases de séquences synthétiques

La présence de symétries dans des bases de séquences de fouille de motifs séquentiels montre que, dans de nombreux cas, plusieurs items sont interchangeables. Ces résultats ouvrent une voie de recherche sur l'interprétation et l'exploitation de telles régularités structurelles.

Prenons, à titre d'illustration, deux bases réelles, à savoir la BIBLE et le Léviathan. Ces deux bases de séquences sont obtenues en convertissant respectivement la Bible et le roman Leviathan de Thomas Hobbes (1651) en bases de séquences. Dans ces bases de séquences, chaque mot correspond à un item. Ainsi, les symétries détectées entre les mots, nous permettraient de comprendre les corrélations entre les mots. Une symétrie entre les mots d'un texte, met en évidence une forte similarité qui mérite d'être

approfondie et analysée. Plus précisément, dans nos expérimentations, nous avons trouvé plusieurs mots symétriques :

— *Bible :* (3912, 3913) (verified, guilty) ; (2947, 2950) (fled, escaped) ; (5243, 5250) (nahshon, gamaliel) ; (6855, 6857) (meadows, gleaned) ; (6722, 6723) (privily, fortify) ; (9886, 9889) (croucheth, spite) ; (6698, 6704) (jether, inclined) ; (6696, 6698) (upbraid, jether).

— *Levithan :* (3176, 3178) (oft, disputed) ; (5002, 5003) (penalties, unwarranted) ; (5350, 5351) (deduce, adviseth) ; (3524, 3525) (formed, founded) ; (7945, 7946) (admonishing, convinced) ; (3558, 3559) (miracle, educated).

La deuxième expérimentation vise à évaluer l'effet de la symétrie sur la recherche de motifs séquentiels fréquents. Les symétries sont intégrées dans l'algorithme GSP comme indiqué dans l'Algorithme 15. Nous avons utilisé l'implémentation de GSP intégrée par Antonio Gomariz Penalver dans SPMF (URL : `http://www.philippe-fournier-viger.com/spmf/index.php?link=download.php`).

Dans la table 5.8, nous fournissons les résultats comparatifs obtenus par GSP et GSP_{Sym} sur des bases de séquences synthétiques. Dans notre expérimentation, nous considérons différentes valeurs du seuil du support minimum λ (en pourcentage du nombre de séquences dans le jeu de données). Pour chaque méthode, nous fournissons le nombre de motifs séquentiels fréquents trouvés ($\#FP$), le temps en secondes nécessaire pour les calculer ($Time$) et l'espace mémoire utilisé par les différentes méthodes. Pour GSP_{Sym}, nous mentionnons également le nombre de motifs séquentiels fréquents déduits par symétrie ($\#FPbySym$). Tout d'abord, les deux algorithmes produisent le même nombre de motifs séquentiels. GSP_{Sym} surpasse GSP à la fois en temps CPU et en espace mémoire (en gras dans le tableau) nécessaire au calcul des motifs séquentiels fréquents. GSP_{Sym} est meilleur en termes de temps CPU (respectivement d'espace mémoire) sur 35 (respectivement 39) instances parmi un total de 48 instances de données. Le synDataset_9 est omis car les algorithmes GSP et GSP_{Sym} n'ont pas pu énumérer tous les motifs séquentiels fréquents (espace mémoire insuffisant).

Dans la troisième expérimentation, notre objectif est de montrer l'effet des symétries sur des bases de séquences synthétiques fortement symétriques. Ces bases de données sont dérivées de l'ensemble symSynDataset_1 obtenu en faisant varier à la fois la taille de la symétrie utilisée (n donné en pourcentage du nombre d'items de la base de données) et la valeur du seuil de support minimum (λ). La table 5.9 démontre que sur les bases de séquences synthétiques fortement symétriques, l'exploitation des symétries permet des réductions significatives à la fois en temps et en espace. Sur toutes les instances des bases de données, GSP_{Sym} est meilleur que GSP. Pour une meilleure vue et lisibilité, les figures 5.2 et 5.3 mettent en évidence les résultats comparatifs entre GSP et GSP_{Sym} sur les deux premières bases de séquences synthétiques et sur deux bases de séquences synthétiques symétriques, tout en faisant varier le seuil de support minimum. Des améliorations similaires sont obtenues sur les autres bases de séquences. Comme nous pouvons l'observer, le gain est plus important pour les faibles valeurs du seuil de support minimum.

En résumé, des symétries sont présentes dans les bases de séquences et le temps pour les détecter est raisonnable. Sur les bases de séquences synthétiques et sur bases de séquences synthétiques fortement symétriques, l'exploitation des symétries améliore les performances des algorithmes d'énumération de motifs séquentiels fréquents (GSP) en temps et en espace.

Dans la dernière expérimentation, nous fournissons l'évaluation expérimentale de notre deuxième approche d'élagage de symétries basée sur le pré-traitement. Comme cette deuxième approche est indépendante de l'algorithme, nous avons considéré dans nos expérimentations plusieurs algorithmes de fouille de motifs séquentiels, notamment SPADE [144], SPAM [13] et PrefixSpan [101]. Tous ces algorithmes sont fournis par Philippe Fournier Viger, et peuvent être trouvés sur le site web de SPMF.

Les premiers résultats décrits dans les tables 5.10 et 5.11 fournissent pour chaque base de séquences,

Synthetic datasets	λ	*GSP*			*GSPSym*			
		#FP	Time(s)	Memory (MB)	#FP	# FPbySym	Time(s)	Memory (MB)
synDataset_1	0.35 %	1	**0.020**	**449.37**	1	0	0.021	484.80
	0.3 %	33	0.219	757.05	33	0	**0.164**	**480.91**
	0.25 %	2123	7.381	1325.55	2123	979	**5.546**	**896.51**
	0.2 %	2214	13.597	1381.55	2214	968	**9.751**	**1034.26**
	0.15 %	287075	1323.32	4255.17	287075	207324	**954.23**	**3756.19**
	0.1 %	799649	13434.88	6299.08	799649	425875	**8394.68**	**3256.32**
synDataset_2	0.35 %	1	**0.007**	621.41	1	0	0.024	**487.83**
	0.3 %	1	**0.018**	**435.38**	1	0	0.023	488.27
	0.25 %	1	0.020	436.89	1	0	**0.018**	**435.19**
	0.2 %	786	5.528	1021.10	786	256	**4.046**	**864.25**
	0.15 %	23706	394.03	**1734.90**	23706	9847	**250.77**	2010.69
	0.1 %	172897	12219.29	4168.55	172897	77279	**9882.24**	**3320.78**
synDataset_3	0.35 %	1	**0.006**	705.59	1	0	0.026	**613.38**
	0.3 %	1	**0.016**	**557.39**	1	0	0.025	614.65
	0.25 %	9	0.135	743.65	9	4	**0.060**	**624.15**
	0.2 %	114	3.821	990.17	114	50	**2.647**	**872.82**
	0.15 %	19714	443.105	1977.87	19714	8631	**370.30**	**1668.14**
	0.1 %	602401	17021.967	5287.97	602401	283248	**10186.73**	**3247.23**
synDataset_4	0.35 %	1	**0.02**	**630.609**	1	0	0.038	754.29
	0.3 %	1	**0.023**	**685.16**	1	0	0.039	739.31
	0.25 %	7	0.181	973.57	7	3	**0.112**	**739.91**
	0.2 %	2222	19.821	1285.69	2222	976	**13.589**	**767.41**
	0.15 %	46818	821.785	1473.21	46818	20058	**639.889**	**937.63**
	0.1 %	277354	23904.609	4480.280	277354	25621	**19547.902**	**1045.84**
synDataset_5	0.35%	8	0.176	945.65	8	3	**0.87**	**775.01**
	0.3 %	8	0.074	706.53	8	3	**0.069**	**623.56**
	0.25%	17	0.399	828.61	17	7	**0.297**	**728.59**
	0.2 %	540	21.755	1326.47	540	235	**15.028**	**1075.98**
	0.15 %	21536	799.013	3515.13	21536	7900	**569.23**	**2999.30**
	0.1 %	462350	30323.89	6195.88	462350	8362	**27225.91**	**3251.18**
synDataset_6	0.35 %	1	**0.02**	743.65	1	0	0.028	**385.69**
	0.3 %	4	**0.059**	**411.51**	4	1	0.060	454.29
	0.25 %	65	0.309	657.20	65	27	**0.247**	**564.87**
	0.2 %	250	3.285	1207.52	250	168	**2.209**	**1011.46**
	0.15 %	13731	239.097	2514.41	13731	653	**193.74**	**1941.67**
	0.1 %	696189	8199.479	7562.85	696189	10457	**6935.749**	**3651.51**
synDataset_7	0.35 %	1	**0.015**	496.57	1	0	0.026	**386.06**
	0.3 %	1	**0.017**	497.70	1	0	0.022	**409.13**
	0.25 %	2	0.045	490.399	2	0	**0.029**	**472.53**
	0.2 %	652	2.727	829.01	652	243	**1.297**	**694.28**
	0.15 %	14389	142.444	2790.50	14389	485	**94.391**	**1254.76**
	0.1 %	145308	7272.784	7078.05	145308	11845	**5911.14**	**6019.24**
synDataset_8	0.35 %	1	**0.019**	**502.51**	1	0	0.020	525.32
	0.3 %	1	**0.02**	**530.17**	1	0	0.025	539.87
	0.25 %	8	0.830	619.069	8	2	**0.658**	**588.64**
	0.2 %	82	1.424	820.64	82	21	**0.982**	**785.98**
	0.15 %	10225	158.834	2954.41	10225	293	**87.4**	**10348.32**
	0.1 %	607316	91162.266	6747.95	607316	17152	**77612.91**	**41799.38**
Statistics			13	9			**35**	**39**

TABLE 5.8 – Extraction de motifs séquentiels fréquents dans les bases de séquences synthétiques : GSP vs $GSPSym$

n	λ	*GSP*			*GSPSym*			
		#FP	Time(s)	Memory (MB)	#FP	# FPbySym	Time(s)	Memory (MB)
50 %	0.4 %	1	0.039	726.245	1	0	**0.033**	**667.098**
	0.35 %	1	0.091	643.583	1	0	**0.085**	**642.717**
	0.3 %	11	0.398	653.470	11	4	**0.319**	**649.247**
	0.25 %	1049	10.478	866.442	1049	405	**8.253**	**811.385**
	0.2 %	1097	26.154	1079.791	1097	489	**21.732**	**1034.331**
	0.15 %	16577	1149.453	919.243	16577	3768	**930.644**	**750.469**
40 %	0.4 %	1	**0.053**	661.121	1	0	0.089	**577.434**
	0.35 %	3	**0.120**	**710.230**	3	0	0.343	753.684
	0.3 %	33	0.441	674.620	33	12	**0.420**	**801.748**
	0.25 %	328	4.886	1008.376	328	95	**3.660**	**745.205**
	0.2 %	381	22.461	884.534	381	145	**14.343**	**794.883**
	0.15 %	73633	1899.242	1590.913	73633	2260	**1519.478**	**1292.657**
30 %	0.4 %	1	**0.089**	**677.447**	1	0	0.161	700.893
	0.35 %	1	**0.027**	735.651	1	0	0.104	**723.505**
	0.3 %	34	**0.609**	794.585	34	9	0.814	**767.908**
	0.25 %	1099	10.834	1187.797	1099	171	**5.744**	**873.872**
	0.2 %	1181	33.543	1486.302	1181	239	**30.817**	**1370.782**
	0.15 %	18253	1199.846	960.382	18253	2513	**643.476**	**927.020**
20 %	0.4 %	1	0.062	703.590	1	0	**0.118**	**695.494**
	0.35 %	1	0.042	**738.567**	1	0	**0.029**	759.472
	0.3 %	22	0.452	**843.741**	22	4	**0.431**	877.023
	0.25 %	1068	11.577	1162.227	1068	42	**4.761**	**1010.259**
	0.2 %	1130	25.255	1451.396	1130	217	**17.674**	**964.017**
	0.15 %	26876	1435.629	1594.096	26876	2871	**1055.668**	**1468.242**
10 %	0.4 %	1	**0.058**	723.074	1	0	0.163	**708.217**
	0.35 %	1	**0.027**	717.203	1	0	0.071	**706.609**
	0.3 %	33	**0.499**	838.831	33	0	0.689	**761.748**
	0.25 %	2123	18.580	1339.070	2123	285	**7.836**	**847.213**
	0.2 %	2211	37.096	1222.337	2211	356	**27.587**	**908.666**
	0.15 %	88703	2092.886	1654.746	3763	373	**1246.012**	**973.367**
Statistics			8	4			**22**	**26**

TABLE 5.9 – Extraction de motifs séquentiels fréquents dans symSynDataset_1 : GSP vs $GSPSym$

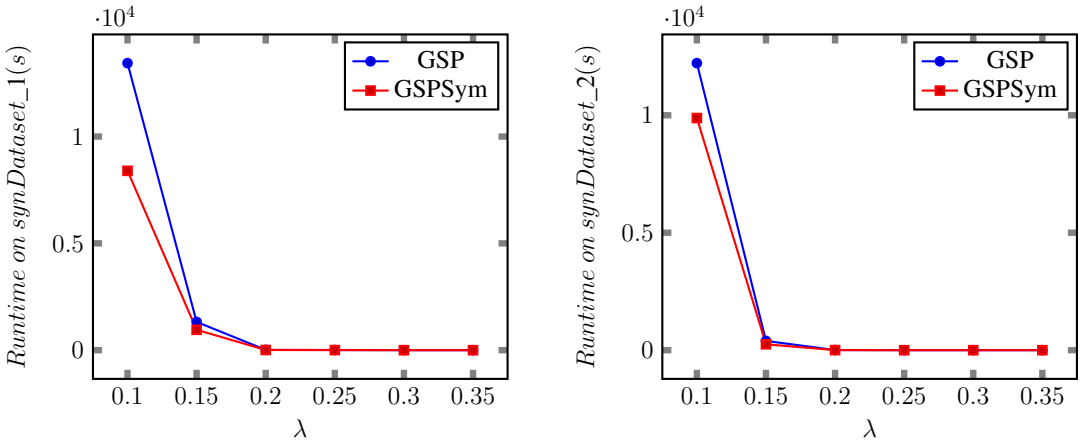


FIGURE 5.2 – Temps d'exécution de différentes bases synthétiques.

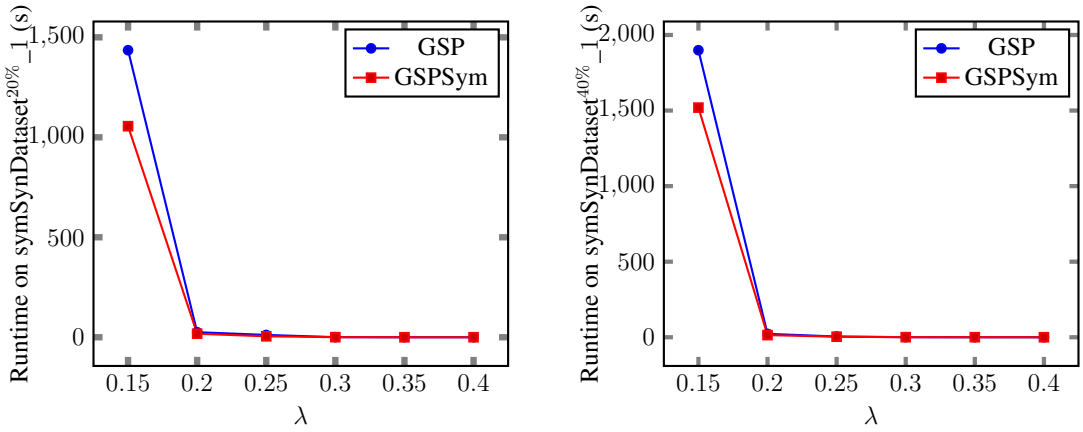


FIGURE 5.3 – Temps d'exécution de différentes bases symSynDataset^n_1.

le nombre total d'items ($|\Omega occ|$) et le nombre d'items supprimés ($|\Omega rem|$) par symétrie de la base de séquences sur les bases de données synthétiques et symétriques. Les résultats montrent que de nombreux items sont supprimés, jusqu'à 753 items sur synDataset_6. Sur les bases de séquences symétriques, le nombre d'items supprimés peut atteindre plusieurs milliers d'items.

Synthetic Datasets	$\|\Delta\|$	$\|\Omega\|$	$\|\Omega occ\|$	$\|\Omega rem\|$	time(s)
synDataset_1	50351	41911	666562	387	123.687
synDataset_2	47784	53137	746942	222	174.784
synDataset_3	47555	62296	854674	120	204.787
synDataset_4	47987	69686	983317	36	212.440
synDataset_5	48466	75476	1100508	41	120.601
synDataset_6	45534	41270	559927	753	73.423
synDataset_7	45451	52551	661072	315	104.954
synDataset_8	46130	61137	770916	175	157.294
synDataset_9	47133	68240	887332	62	183.848

TABLE 5.10 – Résultats de l'élagage de symétries basé sur le prétraitement des bases de séquences synthétiques.

Synthetic Datasets	$\|\Delta\|$	$\|\Omega\|$	$\|\Omega occ\|$	$\|\Omega rem\|$	time(s)
symSynDataset$^{100\%}$_1	100702	41911	1333124	17640	422.880
symSynDataset$^{90\%}$_1	100702	41911	1333124	16205	434.338
symSynDataset$^{80\%}$_1	100702	41911	1333124	14109	450.960
symSynDataset$^{70\%}$_1	100702	41911	1333124	12291	456.839
symSynDataset$^{60\%}$_1	100702	41911	1333124	10844	443.799
symSynDataset$^{50\%}$_1	100702	41911	1333124	8190	584.797
symSynDataset$^{40\%}$_1	100702	41911	1333124	6552	415.418
symSynDataset$^{30\%}$_1	100702	41911	1333124	4736	468.044
symSynDataset$^{20\%}$_1	100702	41911	1333124	3271	446.164
symSynDataset$^{10\%}$_1	100702	41911	1333124	1268	445.265

TABLE 5.11 – Résultats de l'élagage des symétries sur les bases synthétiques symétriques.

Dans les tables 5.12 et 5.13, nous avons comparé plusieurs algorithmes récents dont SPADE, SPAM et PrefixSpan sur les bases de données séquentielles originales et celles obtenues par pré-traitement. Nous observons que les améliorations sont meilleures avec SPAM qu'avec SPADE. PrefixSpan avec un seuil de support minimum faible est meilleur sur les bases de séquences 1, 3, 5, 7 et 8. Dans les différentes tables, nous fournissons dans la dernière ligne ($Statistics$), pour chaque méthode, le nombre d'instances sur lesquelles elle est la plus performante. De même, un échantillon des résultats des tableaux 5.12 et 5.13 sont représentés dans les figures 5.4 et 5.5. Mentionnons également qu'aucun des algorithmes de fouille de motifs séquentiels considérés n'a pu résoudre le dernier ensemble de données synthétiques (synDataset_9).

Synthetic datasets	λ	*SPADE*		*SPADE_PruneSym*		*SPAM*		*SPAM_PruneSym*		*PrefixSpan*		*PrefixSpan_PruneSym*	
		Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)
	0.35 %	0.002	**467.620**	**0.001**	446.623	**0.002**	438.632	**0.002**	**436.335**	0.041	345.099	**0.034**	**343.833**
	0.3 %	0.012	472.633	**0.009**	**465.669**	0.024	**462.127**	**0.011**	473.596	0.179	436.418	**0.049**	**404.329**
	0.25 %	**0.202**	485.123	0.252	**335.823**	0.183	440.251	**0.132**	**422.373**	0.403	371.227	**0.307**	**348.150**
synDataset_1	0.2 %	0.237	**638.858**	**0.167**	546.872	**0.140**	514.701	0.264	**510.323**	0.351	379.298	**0.320**	**363.926**
	0.15 %	4.718	**287.918**	**4.420**	682.797	4.478	**585.429**	**4.123**	594.841	11.182	278.667	**0.847**	**215.613**
	0.1 %	15.457	949.984	**13.670**	**895.397**	33.219	161.179	**28.085**	**151.040**	18.301	525.016	**6.457**	**474.283**
	0.35 %	**0.002**	536.205	**0.002**	**534.591**	**0.001**	**539.840**	**0.001**	541.447	0.062	419.585	**0.037**	**404.185**
	0.3 %	**0.002**	**500.218**	**0.002**	536.931	0.002	**569.761**	**0.001**	577.435	0.056	454.430	**0.044**	**414.339**
	0.25 %	0.002	**537.683**	**0.001**	574.056	0.003	500.047	**0.001**	**494.205**	**0.035**	**433.183**	0.045	445.420
synDataset_2	0.2 %	0.247	**529.048**	**0.203**	560.080	0.137	579.996	**0.089**	**579.920**	0.175	458.020	**0.132**	**450.813**
	0.15 %	1.021	**636.576**	**0.886**	720.084	2.239	587.968	**1.330**	**447.959**	1.394	595.437	**1.108**	**556.764**
	0.1 %	6.108	1099.609	**5.752**	**617.319**	27.474	605.016	**17.275**	**510.524**	12.431	595.214	**5.813**	**420.606**
	0.35 %	0.003	**582.507**	**0.002**	617.642	0.003	620.568	**0.002**	**604.173**	0.062	490.805	**0.041**	**460.052**
	0.3 %	**0.003**	664.545	0.006	**638.727**	**0.002**	**618.542**	0.007	625.561	**0.056**	544.228	0.063	**501.077**
	0.25 %	0.005	**578.911**	**0.003**	586.865	0.008	**662.319**	**0.004**	732.410	0.062	**522.551**	**0.044**	585.071
synDataset_3	0.2 %	0.065	657.719	**0.024**	**651.639**	0.089	**598.703**	**0.065**	604.234	0.113	582.041	**0.087**	**559.767**
	0.15 %	**1.021**	678.204	1.123	**640.941**	**1.868**	**624.557**	1.887	813.30	1.553	**592.359**	**0.727**	600.347
	0.1 %	1.2577	1436.813	**1.2119**	**996.211**	4.117	914.726	**3.001**	**628.832**	624.531	948.865	**7.785**	**673.463**
	0.35 %	**0.002**	720.907	**0.022**	**701.579**	**0.002**	716.527	**0.022**	**699.103**	0.055	543.963	**0.034**	**526.312**
	0.3 %	0.003	**682.541**	**0.002**	783.072	0.003	**704.879**	**0.002**	718.555	**0.049**	**549.636**	**0.049**	598.154
	0.25 %	0.044	**733.752**	**0.003**	822.922	0.009	**741.529**	**0.007**	771.169	0.056	**580.150**	**0.045**	625.325
synDataset_4	0.2 %	0.390	849.079	**0.155**	**729.988**	0.565	769.180	**0.197**	**683.235**	0.485	703.681	**0.241**	**613.057**
	0.15 %	1.402	183.679	**1.225**	**110.326**	3.152	863.329	**2.199**	**826.110**	2.448	767.923	**0.963**	**688.034**
	0.1 %	10.703	1197.185	**9.594**	**922.969**	5.611	756.518	**4.034**	**567.203**	21.286	1050.527	**11.709**	**691.439**
	0.35%	0.005	743.246	**0.004**	**713.192**	**0.005**	**729.957**	**0.005**	786.343	0.079	718.585	**0.060**	**637.379**
	0.3 %	0.004	**841.118**	**0.002**	814.123	**0.028**	825.407	0.04	**789.998**	0.115	714.283	**0.101**	**665.554**
	0.25%	**0.003**	**753.119**	**0.003**	905.397	**0.012**	**732.213**	0.014	813.184	0.394	888.916	**0.236**	**760.184**
synDataset_5	0.2 %	**0.044**	903.293	0.083	**787.919**	0.273	778.030	**0.222**	**729.695**	0.187	775.920	**0.156**	**716.089**
	0.15 %	**1.165**	888.406	1.338	**808.444**	**2.441**	971.234	3.171	**940.417**	1.579	317.814	**1.354**	**209.917**
	0.1 %	**14.274**	1276.955	15.460	**1265.571**	74.817	883.852	**60.030**	**263.272**	16.884	**727.389**	**12.850**	844.168
	0.35 %	**0.001**	395.850	0.002	**375.974**	**0.001**	408.250	0.002	**378.064**	0.064	579.643	**0.054**	**578.033**
	0.3 %	**0.002**	411.048	**0.002**	**377.317**	0.003	415.064	**0.002**	**408.126**	0.055	**597.644**	**0.049**	613.450
	0.25 %	**0.016**	**413.806**	0.035	424.289	**0.014**	**368.479**	0.016	434.376	0.074	**605.529**	**0.063**	619.047
synDataset_6	0.2 %	0.027	**404.060**	**0.019**	427.635	0.049	**374.598**	**0.028**	436.975	0.262	**572.810**	**0.228**	615.508
	0.15 %	0.780	**460.813**	**0.660**	477.987	**1.243**	**434.478**	1.146	509.235	1.649	610.646	**0.924**	**466.658**
	0.1 %	11.527	**530.183**	**10.944**	676.401	23.494	488.940	**18.060**	**358.090**	35.727	1203.301	**13.644**	**962.450**
	0.35 %	**0.002**	**463.340**	**0.002**	478.054	0.002	485.862	**0.001**	**465.689**	**0.040**	466.147	0.083	**380.578**
	0.3 %	0.002	516.357	**0.001**	**493.278**	0.002	504.333	**0.001**	**488.670**	**0.033**	391.505	0.038	**391.269**
	0.25 %	**0.002**	564.035	**0.002**	**469.020**	0.003	533.203	**0.002**	**521.848**	0.043	**393.199**	**0.024**	414.510
synDataset_7	0.2 %	**0.075**	**471.774**	0.079	546.572	0.064	**521.917**	**0.048**	600.713	0.263	529.824	**0.072**	**403.668**
	0.15 %.	**0.691**	**578.227**	0.698	597.523	0.849	551.451	**0.784**	**469.069**	1.081	496.833	**0.402**	**450.424**
	0.1 %	3.959	**669.059**	**3.883**	766.651	16.262	722.355	**14.227**	**646.501**	7.871	551.719	**4.728**	**355.758**
	0.35 %	**0.001**	596.299	**0.001**	**571.556**	**0.002**	**829.453**	0.003	855.248	0.038	**433.588**	**0.029**	480.759
	0.3 %	**0.001**	580.681	**0.001**	**579.466**	**0.022**	**893.709**	0.023	938.676	0.054	**451.974**	**0.039**	502.572
	0.25 %	0.005	643.378	**0.004**	**656.296**	**0.184**	962.747	0.186	**947.426**	**0.043**	473.265	0.049	**447.298**
synDataset_8	0.2 %	**0.012**	**610.162**	**0.012**	618.977	0.098	887.245	**0.089**	**812.064**	**0.069**	562.758	0.164	**434.596**
	0.15 %	0.472	614.542	**0.441**	**608.727**	2.975	**456.827**	**2.780**	757.778	1.008	623.475	**0.472**	**524.082**
	0.1 %	9.307	708.454	**9.063**	**491.857**	53.847	558.078	**42.213**	**498.209**	17.103	640.346	**4.720**	**406.929**
Statistics		20	22	**38**	**25**	16	19	**36**	**29**	7	12	**42**	**36**

TABLE 5.12 – Élagage de symétries basé sur le prétraitement : comparaison d'algorithmes d'extraction de motifs séquentiels sur des bases de séquences synthétiques.

Synthetic datasets	λ	*SPADE*		*SPADE_PruneSym*		*SPAM*		*SPAM_PruneSym*		*PrefixSpan*		*PrefixSpan_PruneSym*	
		Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)	Time (s)	Memory (MB)
100 %	0.35 %	**0.011**	706.495	0.032	**690.168**	**0.010**	730.254	0.025	**709.388**	0.073	**555.832**	**0.056**	611.287
	0.3 %	0.017	**703.956**	**0.014**	736.152	0.046	**711.683**	**0.018**	716.944	0.082	**555.023**	**0.070**	610.388
	0.25 %	**0.127**	739.135	0.145	**706.205**	0.195	702.150	**0.129**	**691.785**	0.296	668.466	**0.161**	**605.359**
	0.2 %	0.217	711.832	**0.180**	**701.082**	0.514	716.527	**0.286**	**713.529**	**0.252**	663.878	0.257	**656.486**
	0.15 %	**0.667**	753.609	0.731	**626.950**	4.967	725.417	**2.732**	**660.020**	1.067	757.500	**0.989**	**691.595**
	0.1 %	**7.251**	525.060	7.854	**363.926**	**77.525**	731.206	49.843	**412.888**	**16.559**	**441.878**	16.627	801.379
	0.095 %	11.039	521.034	**9.068**	**331.722**	87.076	708.773	**78.281**	**704.320**	23.448	1057.247	**21.610**	**439.646**
90 %	0.35 %	**0.002**	787.515	0.003	**763.920**	**0.002**	787.515	0.003	**772.316**	0.059	**582.760**	**0.054**	617.187
	0.3 %	0.012	775.091	**0.010**	**750.505**	**0.023**	**765.915**	0.034	778.083	**0.098**	**608.428**	0.134	664.188
	0.25 %	**0.147**	851.437	0.220	**830.029**	**0.160**	867.270	0.235	**850.768**	0.244	**655.993**	**0.224**	683.748
	0.2 %	0.124	937.979	**0.088**	**737.564**	0.099	**752.769**	**0.077**	769.795	0.404	807.372	**0.197**	**734.710**
	0.15 %	**0.679**	987.686	0.701	**923.456**	3.168	1022.875	**2.844**	**1002.181**	**1.111**	773.254	9.462	**771.946**
	0.1 %	7.152	427.644	**6.516**	**359.559**	53.210	1118.464	**40.340**	**977.353**	19.027	591.040	**15.859**	**318.385**
	0.095 %	9.375	1265.211	**8.930**	**951.437**	55.031	**605.469**	**47.324**	1000.012	25.810	610.954	**22.516**	**465.995**
80 %	0.35 %	**0.003**	814.483	**0.003**	**747.489**	**0.002**	733.577	**0.002**	**726.783**	0.072	**625.949**	**0.070**	638.761
	0.3 %	0.005	798.747	**0.004**	**782.532**	**0.011**	**785.073**	0.038	814.969	**0.060**	649.700	0.066	**639.659**
	0.25 %	**0.035**	**887.598**	0.154	892.464	0.071	890.384	**0.060**	**769.748**	0.147	**605.337**	**0.113**	696.181
	0.2 %	0.223	905.239	**0.077**	**774.223**	**0.128**	**755.378**	0.179	867.454	0.313	715.037	**0.210**	**670.913**
	0.15 %	**0.797**	1090.625	0.956	**805.670**	3.461	1088.028	**2.853**	**927.185**	1.429	957.405	**1.276**	**713.638**
	0.1 %	6.726	1076.764	**6.154**	**1360.963**	55.304	1339.070	**44.503**	**772.606**	19.250	575.651	**16.125**	**425.012**
	0.095 %	9.635	1367.227	**8.338**	**1221.533**	59.647	1359.018	**50.128**	**867.996**	26.001	807.929	**23.286**	**554.692**
70 %	0.35 %	**0.003**	**781.696**	**0.003**	790.893	**0.002**	767.513	**0.002**	**751.662**	**0.055**	**630.028**	0.059	647.769
	0.3 %	0.008	877.239	**0.007**	**750.462**	**0.013**	**860.471**	**0.013**	875.025	0.082	714.642	**0.067**	**641.577**
	0.25 %	**0.018**	**770.280**	0.021	938.305	0.049	931.836	**0.045**	**812.057**	**0.130**	678.392	**0.130**	**641.884**
	0.2 %	**0.018**	842.894	0.022	**771.255**	0.104	**776.635**	**0.098**	939.181	0.215	**739.475**	**0.201**	800.632
	0.15 %	**0.646**	926.712	0.771	**810.502**	2.811	1083.290	**2.065**	**836.900**	**1.408**	293.430	1.521	**284.104**
	0.1 %	6.782	1421.621	**6.337**	**1261.535**	49.887	**659.043**	**40.141**	1094.829	19.139	791.113	**17.517**	**704.420**
	0.095 %	8.844	1609.352	**8.462**	**1269.869**	**53.995**	**867.032**	64.633	1121.578	24.128	872.269	**22.571**	**785.559**
60 %	0.35 %	**0.002**	**802.884**	**0.003**	805.739	**0.003**	818.900	0.003	**816.450**	0.079	718.585	**0.060**	**637.379**
	0.3 %	**0.015**	**876.917**	**0.015**	884.472	**0.023**	901.740	0.024	**718.543**	0.115	714.283	**0.101**	**665.554**
	0.25 %	0.175	**861.909**	**0.112**	959.136	0.151	955.175	**0.124**	**791.756**	0.394	88.916	**0.236**	**760.184**
	0.2 %	0.284	868.657	**0.090**	**793.464**	0.060	1065.151	**0.053**	**905.711**	0.187	775.920	**0.156**	**716.089**
	0.15 %	1.041	1122.076	**0.847**	**800.658**	3.074	1104.578	**2.176**	**952.935**	1.579	317.814	**1.354**	**209.917**
	0.1 %	8.394	**556.544**	**6.574**	1359.377	43.389	1175.642	**35.921**	**901.446**	16.884	**727.389**	**12.850**	844.168
	0.095 %	10.603	**700.436**	**8.868**	1290.630	63.842	**1129.262**	**46.222**	1173.138	25.570	878.256	**21.624**	**790.209**
50 %	0.35 %	**0.002**	860.650	0.003	**788.364**	0.010	**794.021**	**0.002**	817.948	**0.050**	**638.509**	0.055	735.343
	0.3 %	0.008	**800.222**	**0.007**	855.568	**0.013**	**813.841**	0.014	929.211	0.080	716.662	**0.074**	**690.653**
	0.25 %	0.311	**848.996**	**0.236**	908.293	0.248	**811.385**	**0.164**	866.170	**0.267**	**739.130**	0.280	881.620
	0.2 %	**0.091**	974.603	0.238	**944.562**	0.229	**832.861**	**0.154**	857.444	0.305	773.621	**0.263**	**723.735**
	0.15 %	**1.204**	1195.259	1.389	**1187.055**	2.774	**315.616**	**2.729**	434.853	2.286	782.534	**2.066**	**404.172**
	0.1 %	**8.900**	**1110.469**	10.486	1137.663	**11.273**	**1551.556**	55.120	1585.946	21.943	**1125.260**	**21.401**	1262.445
	0.095 %	11.685	1334.601	**10.144**	**1298.052**	80.092	861.246	**78.417**	**830.952**	28.349	1460.766	**25.876**	**1314.453**
40 %	0.35 %	**0.003**	844.106	**0.003**	**812.753**	0.009	**864.666**	**0.007**	869.677	0.071	**686.698**	**0.067**	706.107
	0.3 %	**0.015**	953.307	**0.015**	**897.285**	0.018	876.125	**0.014**	**856.849**	**0.085**	772.436	**0.085**	**672.624**
	0.25 %	**0.040**	**842.837**	0.095	932.461	0.151	**840.139**	**0.121**	912.165	0.174	778.229	**0.162**	**770.269**
	0.2 %	**0.038**	**876.778**	0.107	1006.701	0.282	**930.905**	**0.111**	1028.232	0.260	763.735	**0.229**	**738.660**
	0.15 %	3.905	**1089.596**	**3.209**	1286.746	4.855	1118.273	**4.408**	**1106.679**	6.736	1090.087	**6.058**	**1004.87**
	0.1 %	10.886	446.039	**9.296**	**436.916**	46.561	1189.473	**46.063**	**1134.007**	**25.192**	979.233	25.821	**692.228**
	0.095 %	13.382	1553.918	**13.305**	**1540.954**	66.679	910.113	**53.315**	**896.189**	**33.008**	1116.724	33.792	**735.456**
30 %	0.35 %	**0.003**	863.878	**0.003**	**829.711**	**0.002**	**829.453**	0.003	855.248	**0.057**	**701.251**	0.059	782.080
	0.3 %	**0.014**	951.410	0.015	**929.614**	**0.022**	**893.709**	0.023	938.676	0.089	730.119	**0.076**	**712.149**
	0.25 %	0.227	**975.103**	**0.187**	1064.119	**0.184**	962.747	0.186	**947.426**	0.448	1184.279	**0.396**	**872.590**
	0.2 %	**0.094**	**861.498**	0.146	957.052	0.098	**887.245**	**0.089**	1112.064	**0.242**	913.296	0.322	**705.832**
	0.15 %	1.177	998.787	**0.886**	**941.880**	2.975	1456.827	**2.780**	**1057.778**	2.034	1059.486	**1.924**	**782.750**
	0.1 %	10.994	1506.796	**9.429**	**1501.169**	53.847	1158.078	**42.213**	**998.209**	**21.696**	858.441	22.030	**644.916**
	0.095 %	13.806	1466.037	**11.852**	**1336.127**	81.160	**976.309**	**63.811**	1047.169	**27.532**	990.659	28.642	**742.445**
20 %	0.35 %	**0.003**	935.786	**0.003**	**916.393**	0.026	**891.943**	**0.003**	921.040	**0.059**	750.747	**0.059**	**743.611**
	0.3 %	0.012	**846.464**	**0.008**	865.649	0.016	841.785	**0.015**	**838.448**	**0.069**	801.071	0.097	**736.076**
	0.25 %	0.190	1050.694	**0.181**	**933.527**	0.199	885.194	**0.146**	**840.710**	**0.268**	914.770	0.371	**848.033**
	0.2 %	0.203	**950.167**	**0.159**	1046.336	0.119	1013.439	**0.083**	**933.492**	0.265	879.495	**0.226**	**847.946**
	0.15 %	1.558	1573.521	**1.490**	**1152.428**	3.477	661.649	**2.959**	**584.600**	**3.035**	508.469	3.048	**311.772**
	0.1 %	12.074	991.023	**11.753**	**796.374**	**5.643**	**911.551**	53.618	971.718	22.968	620.627	**22.014**	**610.495**
	0.095 %	15.610	1394.244	**14.249**	**1287.739**	98.708	900.933	**60.528**	**800.556**	31.718	1356.288	**31.007**	**764.042**
10 %	0.35 %	**0.002**	**900.974**	**0.002**	929.835	0.003	945.587	**0.002**	**893.630**	0.065	**723.567**	**0.060**	778.550
	0.3 %	0.014	**870.632**	**0.011**	1016.572	0.061	**898.527**	**0.025**	1028.506	**0.105**	782.859	0.093	**746.272**
	0.25 %	0.282	957.376	**0.279**	**890.984**	0.304	976.407	**0.300**	**937.179**	**0.387**	919.278	0.607	**840.953**
	0.2 %	0.247	1059.361	**0.199**	**901.614**	0.203	1026.445	**0.194**	**873.584**	**0.320**	954.454	0.531	**805.079**
	0.15 %	4.175	1174.194	**2.738**	**1071.313**	5.643	911.551	**4.567**	**874.583**	8.596	1331.308	**6.596**	**989.471**
	0.1 %	17.964	792.592	**16.631**	**411.137**	66.929	**978.480**	**49.875**	1054.168	33.308	339.007	**23.579**	**139.107**
	0.095 %	21.326	**672.125**	**20.430**	1104.222	69.341	961.205	**57.760**	**904.509**	44.486	551.097	**35.345**	**499.454**
Statistics		29	23	**50**	**48**	19	28	**54**	**42**	23	17	**49**	**53**

TABLE 5.13 – Élagage de symétries basé sur le prétraitement : comparaison d'algorithmes d'extraction de motifs séquentiels sur symSynDataset_1

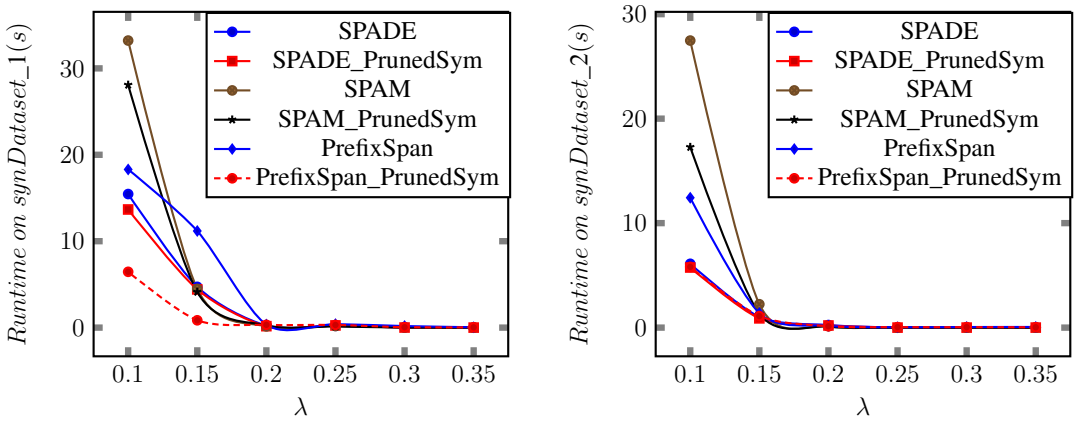


FIGURE 5.4 – Élagage de symétries basé sur le prétraitement de différentes bases de séquences synthétiques.

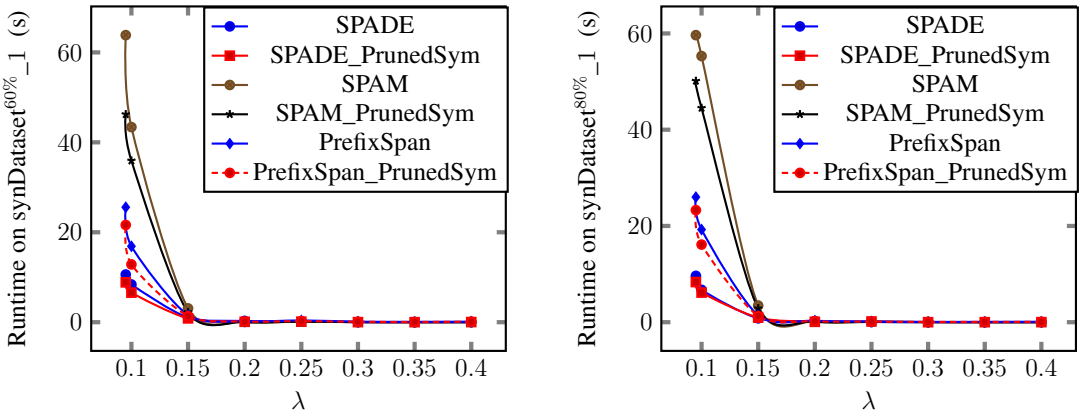


FIGURE 5.5 – Élagage de symétries basé sur le prétraitement de symSynDataset^n_1.

5.5 Conclusion

Dans ce chapitre, nous avons étendu le cadre de la symétrie aux tâches d'extraction de motifs séquentiels. Après avoir formulé le concept de symétries dans les bases de données séquentielles, nous avons montré comment de telles symétries peuvent être détectées grâce à une réduction au problème de la recherche des automorphismes de graphes colorés dirigés. Ensuite, nous avons décrit deux approches d'élagage des symétries. Dans la première, les symétries ont été exploitées pour améliorer les algorithmes de fouille de motifs séquentiels, notamment les algorithmes de type Apriori. La seconde approche vise à éliminer les symétries lors d'une étape de pré-traitement en réécrivant la base de données séquentielle.

Les résultats expérimentaux menés sur plusieurs bases séquentielles bien connues montrent que ces bases contiennent plusieurs symétries révélant des régularités intéressantes qui peuvent être considérées comme un nouveau type de connaissance. Les résultats confirment le grand intérêt de l'utilisation des symétries pour la réduction de l'espace de recherche. Ceci est particulièrement pertinent pour toutes les tâches d'énumération de motifs.

Discussion et conclusion générale

Dans cette thèse nous nous sommes intéressés à une tâche fondamentale en fouille de données, qui est la fouille des motifs fréquents. Cette tâche est appliquée dans plusieurs domaines importants. Il s'agit de trouver des motifs pertinents et utiles à partir de différents types de bases de données. Dans cette thèse, nous avons focalisé notre attention sur les bases de transactions pour la fouille des itemsets ainsi que sur les bases de séquences pour la fouille des séquences. Notre contribution pour l'extraction des itemsets consiste principalement à améliorer une approche déclarative basée sur la satisfiabilité propositionnelle par la compression d'un encodage SAT pour la fouille des itemsets. Tandis que notre deuxième contribution dans le domaine de la fouille de séquences a porté sur la détection des symétries dans les séquences et leurs exploitations pour améliorer le processus d'extraction de séquences.

Dans le cadre de la fouille des itemsets, nous avons commencé par montrer comment la compression de l'encodage SAT pour la fouille des itemsets peut être vue comme un problème de compression de matrice booléenne. Puis, nous avons adapté un encodage SAT pour la fouille des itemsets en introduisant de nouvelles variables et contraintes. Afin de réduire la taille de notre encodage, nous avons proposé de réordonner les transactions de la base originale en utilisant une approche gloutonne. Notre encodage permet de compresser l'encodage par réduction du nombre de clauses générées. Des expérimentations montrent l'intérêt de notre approche proposée.

Dans le cadre de la fouille des séquences, nous avons proposé une méthode de détection des symétries dans les bases de séquences. Ensuite, nous avons montré comment le problème de la fouille des séquences peut être amélioré en exploitant les symétries. Premièrement, nous avons intégré ces symétries détectées dans un algorithme de type Apriori. Deuxièmement, nous avons proposé de faire un prétraitement de la base de séquences en nettoyant les symétries pour optimiser l'espace de recherche. Nos contributions montrent que l'exploitation des symétries dans la fouille de séquences permet de réduire l'espace de recherche et donc les temps de calculs.

Perspectives

Les travaux effectués pendant cette thèse ouvrent de nombreuses perspectives. Tout d'abord l'approche de compression de l'encodage basé sur la satisfiabilité propositionnelle proposée pour la fouille des itemsets est une approche générale qui a réduit significativement la taille de l'encodage et le nombre de clauses générées. Cette optimisation a montré une amélioration des performances et ouvre la voie à une extension de notre approche de compression à d'autres tâches de fouille de données comme la fouille des règles d'association ou des séquences. Une deuxième perspective de nos travaux concerne l'approche gloutonne utilisée pour réorganiser la base de transactions. Trouver d'autres heuristiques d'ordonnancement peut aider à réduire encore d'avantage la taille de l'encodage.

Dans notre seconde contribution, nous avons proposé une méthode pour rechercher des symétries globales dans les bases de séquences en transformant la base de séquences en un graphe orienté coloré. Ces symétries sont énumérées en calculant les automorphismes du graphe obtenu. Il sera pertinent de trouver des symétries locales pendant la génération des candidats. En effet, pendant la génération,

la structure de table de séquence change et détecter des symétries sur la nouvelle table permettrait la découverte de nouvelles symétries non présentes dans la table originale.

Une autre perspective de notre approche d'extraction de motifs séquentiels concerne l'exploitation des symétries détectées pour l'élagage de l'espace de recherche. En effet, dans le travail effectué, les symétries ont été intégrées dans un algorithme de type Apriori. Il sera intéressant d'étudier la possibilité d'intégrer les symétries pour l'extraction de motifs séquentiels dans les algorithmes de recherche en profondeur et de les comparer avec l'algorithme GSPSym.

Bibliographie

[1] R. AGRAWAL et R. SRIKANT. “Mining sequential patterns”. In : *Proceedings of the Eleventh International Conference on Data Engineering*. 1995, p. 3-14.

[2] Rakesh AGRAWAL, Tomasz IMIELI&#324 ;SKI et Arun SWAMI. “Mining association rules between sets of items in large databases”. In : *SIGMOD ’93 : Proceedings of the 1993 ACM SIGMOD international conference on Management of data*. Washington, D.C., United States : ACM Press, 1993, p. 207-216. ISBN : 0-89791-592-5.

[3] Rakesh AGRAWAL et Ramakrishnan SRIKANT. “Fast algorithms for mining association rules”. In : (1998), p. 580-592.

[4] Chowdhury Farhan AHMED, Syed Khairuzzaman TANBEER et Byeong-Soo JEONG. “A Novel Approach for Mining High-Utility Sequential Patterns in Sequence Databases”. In : *ETRI journal* 32.5 (2010), p. 676-686.

[5] Sheldon B. AKERS. “Binary decision diagrams”. In : *IEEE Transactions on computers* 27.06 (1978), p. 509-516.

[6] Giulio ALIBERTI et al. “EXPEDITE : EXPress closED ITemset Enumeration”. In : *Expert Syst. Appl.* 42.8 (2015), p. 3933-3944.

[7] Oznur Kirmemis ALKAN et Pinar KARAGOZ. “CRoM and HuspExt : Improving efficiency of high utility sequential pattern extraction”. In : *IEEE Transactions on Knowledge and Data Engineering* 27.10 (2015), p. 2645-2657.

[8] Fadi A. ALOUL et al. “Solving Difficult Instances of Boolean Satisfiability in the Presence of Symmetry”. In : *Transactions on Computer Aided Design* (2003).

[9] Gilles AUDEMARD et Laurent SIMON. “Glucose : a solver that predicts learnt clauses quality”. In : *SAT Competition* (2009), p. 7-8.

[10] Gilles AUDEMARD et Laurent SIMON. “Predicting learnt clauses quality in modern SAT solvers”. In : *Twenty-first International Joint Conference on Artificial Intelligence*. 2009.

[11] Gilles AUDEMARD et al. “A SAT based approach for solving formulas over boolean and linear mathematical propositions”. In : *International Conference on Automated Deduction*. Springer. 2002, p. 195-210.

[12] Gilles AUDEMARD et al. “On freezing and reactivating learnt clauses”. In : *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2011, p. 188-200.

[13] J. AYRES et al. *Sequential pattern mining using a bitmap representation*. 2002.

[14] Hector J. LEVESQUE BART SELMAN et David MITCHELL. “Gsat : A new method for solving hard satisfiability problems”. In : *Proceedings of the Tenth American National Conference on Artificial Intelligence (AAAI’92)*. 1992, p. 440-446.

[15] Yves BASTIDE et al. “Mining Frequent Patterns with Counting Inference”. In : *SIGKDD Explor. Newsl.* 2.2 (déc. 2000), p. 66-75. ISSN : 1931-0145.

[16] Roberto J. BAYARDO. “Efficiently Mining Long Patterns from Databases”. In : *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’98. Seattle, Washington, USA : Association for Computing Machinery, 1998, p. 85-93. ISBN : 0897919955.

[17] Goerge BOOLE. “Les lois de la pensée”. In : Mathesis, 1854.

[18] Kellogg S. BOOTH et George S. LUEKER. “Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms”. In : *J. Comput. Syst. Sci.* 13.3 (1976), p. 335-379.

[19] Abdelhamid BOUDANE et al. “Enumerating Non-redundant Association Rules Using Satisfiability”. In : *Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part I*. Sous la dir. de Jinho KIM et al. T. 10234. Lecture Notes in Computer Science. 2017, p. 824-836.

[20] D. BURDICK, M. CALIMLIM et J. GEHRKE. “MAFIA : a maximal frequent itemset algorithm for transactional databases”. In : *Proceedings 17th International Conference on Data Engineering*. 2001, p. 443-452.

[21] Toon CALDERS, Nele DEXTERS et Bart GOETHALS. “Mining Frequent Itemsets in a Stream”. In : *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*. IEEE Computer Society, 2007, p. 83-92.

[22] Joong Hyuk CHANG. “Mining weighted sequential patterns in a sequence database with a time-interval weight”. In : *Knowledge-Based Systems* 24.1 (2011), p. 1-9.

[23] Joong Hyuk CHANG et Won Suk LEE. “Efficient mining method for retrieving sequential patterns over online data streams”. In : *Journal of Information Science* 31.5 (2005), p. 420-432.

[24] Joong Hyuk CHANG et Won Suk LEE. “Finding recent frequent itemsets adaptively over online data streams”. In : *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*. Sous la dir. de Lise GETOOR et al. ACM, 2003, p. 487-492.

[25] Lei CHANG et al. “Seqstream : Mining closed sequential patterns over stream sliding windows”. In : *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, p. 83-92.

[26] Pankaj CHAUHAN, Edmund Melson CLARKE et Daniel KROENING. *Using SAT based image computation for reachability analysis*. Citeseer, 2003.

[27] Yun CHI et al. “Catch the moment : maintaining closed frequent itemsets over a data stream sliding window”. In : *Knowl. Inf. Syst.* 10.3 (2006), p. 265-294.

[28] Stephen A. COOK. “The Complexity of Theorem-Proving Procedures”. In : *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC ’71. Shaker Heights, Ohio, USA : Association for Computing Machinery, 1971, p. 151-158. ISBN : 9781450374644.

[29] Martin DAVIS, George LOGEMANN et Donald LOVELAND. “A machine program for theorem-proving”. In : *Communications of the ACM* 5.7 (1962), p. 394-397.

[30] Martin DAVIS et Hilary PUTNAM. “A computing procedure for quantification theory”. In : *Journal of the ACM (JACM)* 7.3 (1960), p. 201-215.

[31] Imen Ouled DLALA et al. “A Parallel SAT-Based Framework for Closed Frequent Itemsets Mining”. In : *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*. 2018, p. 570-587.

[32] William F DOWLING et Jean H GALLIER. “Linear-time algorithms for testing the satisfiability of propositional Horn formulae”. In : *The Journal of Logic Programming* 1.3 (1984), p. 267-284.

[33] Yiheng DUAN et al. “Detective : Automatically identify and analyze malware processes in forensic scenarios via DLLs”. In : *2015 IEEE International Conference on Communications (ICC)*. IEEE. 2015, p. 5691-5696.

[34] Niklas EEN. “MiniSat : A SAT solver with conflict-clause minimization”. In : *Proc. SAT-05 : 8th Int. Conf. on Theory and Applications of Satisfiability Testing*. 2005, p. 502-518.

[35] Niklas EÉN et Niklas SÖRENSSON. “An extensible SAT-solver”. In : *International conference on theory and applications of satisfiability testing*. Springer. 2003, p. 502-518.

[36] Zahra FARZANYAR, Mohammad Reza KANGAVARI et Nick CERCONE. “Max-FISM : Mining (recently) maximal frequent itemsets over data streams using the sliding window model”. In : *Comput. Math. Appl.* 64.6 (2012), p. 1706-1718.

[37] Basura FERNANDO, Elisa FROMONT et Tinne TUYTELAARS. “Effective use of frequent itemset mining for image classification”. In : *European conference on computer vision*. Springer. 2012, p. 214-227.

[38] Philippe FOURNIER-VIGER, Roger NKAMBOU et Engelbert Mephu NGUIFO. “A Knowledge Discovery Framework for Learning Task Models from User Interactions in Intelligent Tutoring Systems”. In : *CoRR* abs/0901.4761 (2009).

[39] Philippe FOURNIER-VIGER, Cheng-Wei WU et Vincent S. TSENG. “Mining Maximal Sequential Patterns without Candidate Maintenance”. In : *ADMA (1)*. T. 8346. Lecture Notes in Computer Science. Springer, 2013, p. 169-180.

[40] Philippe FOURNIER-VIGER et al. “A survey of itemset mining”. In : *Wiley Interdisciplinary Reviews : Data Mining and Knowledge Discovery* 7 (2017).

[41] Philippe FOURNIER-VIGER et al. “Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information”. In : *PAKDD (1)*. T. 8443. Lecture Notes in Computer Science. Springer, 2014, p. 40-52.

[42] Philippe FOURNIER-VIGER et al. “FHM : Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning”. In : *Foundations of Intelligent Systems - 21st International Symposium, ISMIS 2014, Roskilde, Denmark, June 25-27, 2014. Proceedings*. Sous la dir. de Troels ANDREASEN et al. T. 8502. Lecture Notes in Computer Science. Springer, 2014, p. 83-92.

[43] Philippe FOURNIER-VIGER et al. “TKS : efficient mining of top-k sequential patterns”. In : *International Conference on Advanced Data Mining and Applications*. Springer. 2013, p. 109-120.

[44] Philippe FOURNIER-VIGER et al. “VGEN : Fast Vertical Mining of Sequential Generator Patterns”. In : *DaWaK*. T. 8646. Lecture Notes in Computer Science. Springer, 2014, p. 476-488.

[45] Philippe FOURNIER-VIGER et al. “VMSP : Efficient Vertical Mining of Maximal Sequential Patterns”. In : *Canadian Conference on AI*. T. 8436. Lecture Notes in Computer Science. Springer, 2014, p. 83-94.

[46] Fabio FUMAROLA et al. “CloFAST : closed sequential pattern mining using sparse and vertical id-lists”. In : *Knowl. Inf. Syst.* 48.2 (2016), p. 429-463.

[47] Zvi GALIL. “On the complexity of regular resolution and the Davis-Putnam procedure”. In : *Theoretical Computer Science* 4.1 (1977), p. 23-46. ISSN : 0304-3975.

[48] Chuancong GAO et al. “Efficient mining of frequent sequence generators”. In : *WWW*. ACM, 2008, p. 1051-1052.

[49] René Arnulfo GARCIA-HERNÁNDEZ, José Francisco Martınez TRINIDAD et Jesús Ariel CARRASCO-OCHOA. “A New Algorithm for Fast Discovery of Maximal Sequential Patterns in a Document Collection”. In : *CICLing*. T. 3878. Lecture Notes in Computer Science. Springer, 2006, p. 514-523.

[50] M. R. GAREY, David S. JOHNSON et Larry J. STOCKMEYER. “Some Simplified NP-Complete Graph Problems”. In : *Theor. Comput. Sci.* 1.3 (1976), p. 237-267.

[51] Antonio GOMARIZ et al. “ClaSP : An Efficient Algorithm for Mining Frequent Closed Sequences”. In : *PAKDD (1)*. T. 7818. Lecture Notes in Computer Science. Springer, 2013, p. 50-61.

[52] Carla P GOMES, Bart SELMAN, Henry KAUTZ et al. “Boosting combinatorial search through randomization”. In : *AAAI/IAAI* 98 (1998), p. 431-437.

[53] Carla P GOMES et al. “Heavy-tailed phenomena in satisfiability and constraint satisfaction problems”. In : *Journal of automated reasoning* 24.1 (2000), p. 67-100.

[54] K. GOUDA et M.J. ZAKI. “Efficiently mining maximal frequent itemsets”. In : *Proceedings 2001 IEEE International Conference on Data Mining*. 2001, p. 163-170.

[55] En-Zheng GUAN et al. “Mining Maximal Sequential Patterns”. In : *2005 International Conference on Neural Networks and Brain*. T. 1. 2005, p. 525-528.

[56] Tias GUNS, Siegfried NIJSSEN et Luc De RAEDT. “Itemset mining : A constraint programming perspective”. In : *Artif. Intell.* 175.12-13 (2011), p. 1951-1983.

[57] Youssef HAMADI, Said JABBOUR et Lakhdar SAIS. “Learning for dynamic subsumption”. In : *International Journal on Artificial Intelligence Tools* 19.04 (2010), p. 511-529.

[58] Jiawei HAN et Jian PEI. “Mining Frequent Patterns by Pattern-Growth : Methodology and Implications”. In : *SIGKDD Explor.* 2.2 (2000), p. 14-20.

[59] Jiawei HAN et al. “FreeSpan : frequent pattern-projected sequential pattern mining”. In : *KDD*. 2000, p. 355-359.

[60] Jiawei HAN et al. “Mining Frequent Patterns without Candidate Generation : A Frequent-Pattern Tree Approach”. In : *Data Mining and Knowledge Discovery* 8.1 (jan. 2004), p. 53-87.

[61] Markus HEGLAND. “The Apriori Algorithm–a Tutorial”. In : 11 (jan. 2008). DOI : 10.1142/9789812709066_0006.

[62] Yu HIRATE et Hayato YAMANA. “Generalized Sequential Pattern Mining with Item Intervals.” In : *J. Comput.* 1.3 (2006), p. 51-60.

[63] Chin-Chuan HO et al. “Incremental Mining of Sequential Patterns over a Stream Sliding Window”. In : *ICDM Workshops*. IEEE Computer Society, 2006, p. 677-681.

[64] Joshua HO, Lior LUKOV et Sanjay CHAWLA. “Sequential pattern mining with constraints on large protein databases”. In : *Proceedings of the 12th international conference on management of data (COMAD)*. 2005, p. 89-100.

[65] Ya-Han HU et Yen-Liang CHEN. “Mining association rules with multiple minimum supports : a new mining algorithm and a support tuning mechanism”. In : *Decision Support Systems* 42.1 (2006), p. 1-24. ISSN : 0167-9236.

[66] Jinbo HUANG et al. “The Effect of Restarts on the Efficiency of Clause Learning.” In : *IJCAI*. T. 7. 2007, p. 2318-2323.

[67] Said JABBOUR, Lakhdar SAIS et Yakoub SALHI. “On SAT Models Enumeration in Itemset Mining”. In : *CoRR* abs/1506.02561 (2015).

[68] Said JABBOUR, Lakhdar SAIS et Yakoub SALHI. “The Top-k Frequent Closed Itemset Mining Using Top-k SAT Problem”. In : *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*. Sous la dir. d’Hendrik BLOCKEEL et al. T. 8190. Lecture Notes in Computer Science. Springer, 2013, p. 403-418.

[69] Said JABBOUR et al. “Extending modern SAT solvers for models enumeration”. In : *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*. IEEE. 2014, p. 803-810.

[70] Said JABBOUR et al. “On Maximal Frequent Itemsets Mining with Constraints”. In : *CP*. T. 11008. Lecture Notes in Computer Science. Springer, 2018, p. 554-569.

[71] Said JABBOUR et al. “Revisiting the learned clauses database reduction strategies”. In : *International Journal on Artificial Intelligence Tools* 27.08 (2018), p. 1850033.

[72] Said JABBOUR et al. “Symmetries in Itemset Mining”. In : *ECAI*. T. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, p. 432-437.

[73] HoonSang JIN, HyoJung HAN et Fabio SOMENZI. “Efficient conflict analysis for finding all satisfying assignments of a Boolean circuit”. In : *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2005, p. 287-300.

[74] Tommi A. JUNTTILA et Petteri KASKI. “Engineering an Efficient Canonical Labeling Tool for Large and Sparse Graphs”. In : *ALENEX*. SIAM, 2007.

[75] Hyeong-Ju KANG et In-Cheol PARK. “SAT-based unbounded symbolic model checking”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.2 (2005), p. 129-140.

[76] R. Uday KIRAN et P. Krishna REDDY. “Novel Techniques to Reduce Search Space in Multiple Minimum Supports-Based Frequent Pattern Mining Algorithms”. In : *Proceedings of the 14th International Conference on Extending Database Technology*. EDBT/ICDT ’11. Uppsala, Sweden : Association for Computing Machinery, 2011, p. 11-20. ISBN : 9781450305280.

[77] Jia-Ling KOH et Shui-Feng SHIEH. “An Efficient Approach for Maintaining Association Rules Based on Adjusting FP-Tree Structures1”. In : *Database Systems for Advances Applications, 9th International Conference, DASFAA 2004, Jeju Island, Korea, March 17-19, 2004, Proceedings*. Sous la dir. d’Yoon-Joon LEE et al. T. 2973. Lecture Notes in Computer Science. Springer, 2004, p. 417-424.

[78] Yun Sing KOH et Sri Devi RAVANA. “Unsupervised Rare Pattern Mining : A Survey”. In : *ACM Trans. Knowl. Discov. Data* 10.4 (mai 2016). ISSN : 1556-4681.

[79] Yun Sing KOH et Nathan ROUNTREE. “Finding Sporadic Rules Using Apriori-Inverse”. In : *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings*. Sous la dir. de Tu Bao HO, David Wai-Lok CHEUNG et Huan LIU. T. 3518. Lecture Notes in Computer Science. Springer, 2005, p. 97-106.

[80] Guo-Cheng LAN et al. “Applying the maximum utility measure in high utility sequential pattern mining”. In : *Expert Systems with Applications* 41.11 (2014), p. 5071-5081.

[81] Nadjib LAZAAR et al. “A Global Constraint for Closed Frequent Pattern Mining”. In : *CP*. T. 9892. Lecture Notes in Computer Science. Springer, 2016, p. 333-349.

[82] Carson Kai-Sang LEUNG et al. “CanTree : a canonical-order tree for incremental frequent-pattern mining”. In : *Knowl. Inf. Syst.* 11.3 (2007), p. 287-311.

[83] Bin LI, M.S. HSIAO et Shuo SHENG. “A novel SAT all-solutions solver for efficient preimage computation”. In : *Proceedings Design, Automation and Test in Europe Conference and Exhibition*. T. 1. 2004, 272-277 Vol.1.

[84] Chun-Wei LIN, Tzung-Pei HONG et Wen-Hsiang LU. “An effective tree structure for mining high utility itemsets”. In : *Expert Syst. Appl.* 38.6 (2011), p. 7419-7424.

[85] Chun-Wei LIN, Tzung-Pei HONG et Wen-Hsiang LU. “The Pre-FUFP algorithm for incremental mining”. In : *Expert Syst. Appl.* 36.5 (2009), p. 9498-9505.

[86] Ying Chun LIN, Cheng-Wei WU et Vincent S. TSENG. “Mining High Utility Itemsets in Big Data”. In : *Advances in Knowledge Discovery and Data Mining - 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19-22, 2015, Proceedings, Part II*. Sous la dir. de Tru H. CAO et al. T. 9078. Lecture Notes in Computer Science. Springer, 2015, p. 649-661.

[87] Bing LIU, Wynne HSU et Yiming MA. “Mining Association Rules with Multiple Minimum Supports”. In : *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’99. San Diego, California, USA : Association for Computing Machinery, 1999, p. 337-341. ISBN : 1581131437.

[88] Mengchi LIU et Jun-Feng QU. “Mining high utility itemsets without candidate generation”. In : *21st ACM International Conference on Information and Knowledge Management, CIKM’12, Maui, HI, USA, October 29 - November 02, 2012*. Sous la dir. de Xue-wen CHEN et al. ACM, 2012, p. 55-64.

[89] Ying LIU, Wei-keng LIAO et Alok N. CHOUDHARY. “A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets”. In : *Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, Hanoi, Vietnam, May 18-20, 2005, Proceedings*. Sous la dir. de Tu Bao HO, David Wai-Lok CHEUNG et Huan LIU. T. 3518. Lecture Notes in Computer Science. Springer, 2005, p. 689-695.

[90] David LO, Siau-Cheng KHOO et Jinyan LI. “Mining and Ranking Generators of Sequential Patterns”. In : *SDM*. SIAM, 2008, p. 553-564.

[91] S LU et C LI. “AprioriAdjust : An efficient algorithm for discovering the maximum sequential patterns”. In : *Proc. Intern. Workshop knowl. Grid and grid intell*. 2004.

[92] Claudio LUCCHESE, Salvatore ORLANDO et Raffaele PEREGO. “Fast and Memory Efficient Mining of Frequent Closed Itemsets”. In : *IEEE Trans. Knowl. Data Eng.* 18.1 (2006), p. 21-36.

[93] Ken L MCMILLAN. “Applying SAT methods in unbounded symbolic model checking”. In : *International Conference on Computer Aided Verification*. Springer. 2002, p. 250-264.

[94] A MORGADO et J MARQUES-SILVA. *Algorithms for propositional model enumeration and counting*. Rapp. tech. Citeseer, 2005.

[95] Matthew W. MOSKEWICZ et al. “Chaff : Engineering an Efficient SAT Solver”. In : *Proceedings of the 38th Annual Design Automation Conference*. DAC ’01. Las Vegas, Nevada, USA : Association for Computing Machinery, 2001, p. 530-535. ISBN : 1581132972.

[96] Stefan NAULAERTS et al. “A primer to frequent itemset mining for bioinformatics”. In : *Briefings in Bioinformatics* 16.2 (oct. 2013), p. 216-231. ISSN : 1467-5463.

[97] Jong Soo PARK, Ming-Syan CHEN et Philip S. YU. “An Effective Hash Based Algorithm for Mining Association Rules”. In : *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995*. Sous la dir. de Michael J. CAREY et Donovan A. SCHNEIDER. ACM Press, 1995, p. 175-186.

[98] Nicolas PASQUIER et al. “Discovering Frequent Closed Itemsets for Association Rules”. In : *Database Theory - ICDT ’99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*. Sous la dir. de Catriel BEERI et Peter BUNEMAN. T. 1540. Lecture Notes in Computer Science. Springer, 1999, p. 398-416.

[99] Jian PEI, Jiawei HAN et Wei WANG. “Constraint-based sequential pattern mining : the pattern-growth methods”. In : *Journal of Intelligent Information Systems* 28.2 (2007), p. 133-160.

[100] Jian PEI et al. “H-mine : hyper-structure mining of frequent patterns in large databases”. In : *Proceedings 2001 IEEE International Conference on Data Mining*. 2001, p. 441-448.

[101] Jian PEI et al. “Mining sequential patterns by pattern-growth : the PrefixSpan approach”. In : *IEEE Transactions on Knowledge and Data Engineering* 16.11 (2004), p. 1424-1440.

[102] Jian PEI et al. “PrefixSpan, : mining sequential patterns efficiently by prefix-projected pattern growth”. In : *Proceedings 17th International Conference on Data Engineering*. 2001, p. 215-224.

[103] Helen PINTO et al. “Multi-Dimensional Sequential Pattern Mining”. In : *CIKM*. ACM, 2001, p. 81-88.

[104] Knot PIPATSRISAWAT et Adnan DARWICHE. “On the power of clause-learning SAT solvers with restarts”. In : *International Conference on Principles and Practice of Constraint Programming*. Springer. 2009, p. 654-668.

[105] Luc De RAEDT, Tias GUNS et Siegfried NIJSSEN. “Constraint programming for itemset mining”. In : *KDD*. ACM, 2008, p. 204-212.

[106] Chedy RAISSI, Pascal PONCELET et Maguelonne TEISSEIRE. “SPEED : mining maxirnal sequential patterns over data strearns”. In : *2006 3rd International IEEE Conference Intelligent Systems*. IEEE. 2006, p. 546-552.

[107] K RAVI et F SOMENZI. “Minimal satisfying assignments for conjunctive normal formulae”. In : *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2004.

[108] Jia-Dong REN, Jing YANG et Yan LI. “Mining weighted closed sequential patterns in large databases”. In : *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*. T. 5. IEEE. 2008, p. 640-644.

[109] J. A. ROBINSON. “A Machine-Oriented Logic Based on the Resolution Principle”. In : *J. ACM* 12.1 (jan. 1965), p. 23-41. ISSN : 0004-5411.

[110] Vadim RYVCHIN et Ofer STRICHMAN. “Local restarts”. In : *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2008, p. 271-276.

[111] Ashok SAVASERE, Edward OMIECINSKI et Shamkant B. NAVATHE. “An Efficient Algorithm for Mining Association Rules in Large Databases”. In : *VLDB’95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*. Sous la dir. d’Umeshwar DAYAL, Peter M. D. GRAY et Shojiro NISHIO. Morgan Kaufmann, 1995, p. 432-444.

[112] Pierre SCHAUS, John O. R. AOGA et Tias GUNS. "CoverSize : A Global Constraint for Frequency-Based Itemset Mining". In : *International Conference on Principles and Practice of Constraint Programming*. 2017, p. 529-546.

[113] M. SENO et G. KARYPIS. "SLPMiner : an algorithm for finding frequent sequential patterns using length-decreasing support constraint". In : *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* 2002, p. 418-425.

[114] Claude E SHANNON. "A symbolic analysis of relay and switching circuits". In : *Electrical Engineering* 57.12 (1938), p. 713-723.

[115] Shuo SHENG et Michael HSIAO. "Efficient preimage computation using a novel success-driven atpg". In : *2003 Design, Automation and Test in Europe Conference and Exhibition*. IEEE. 2003, p. 822-827.

[116] Se Jung SHIN, Dae Su LEE et Won Suk LEE. "CP-tree : An adaptive synopsis structure for compressing frequent itemsets over online data streams". In : *Inf. Sci.* 278 (2014), p. 559-576.

[117] Panida SONGRAM, Veera BOONJING et Sarun INTAKOSUM. "Closed Multidimensional Sequential Pattern Mining". In : *ITNG*. IEEE Computer Society, 2006, p. 512-517.

[118] Arnaud SOULET et François RIOULT. "Efficiently Depth-First Minimal Pattern Mining". In : *Advances in Knowledge Discovery and Data Mining*. Sous la dir. de Vincent S. TSENG et al. Cham : Springer International Publishing, 2014, p. 28-39.

[119] Ramakrishnan SRIKANT et Rakesh AGRAWAL. "Mining Sequential Patterns : Generalizations and Performance Improvements". In : *Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings*. T. 1057. Lecture Notes in Computer Science. Springer, 1996, p. 3-17.

[120] Ramakrishnan SRIKANT et Rakesh AGRAWAL. "Mining sequential patterns : Generalizations and performance improvements". In : *Advances in Database Technology — EDBT '96*. Sous la dir. de Peter APERS, Mokrane BOUZEGHOUB et Georges GARDARIN. Berlin, Heidelberg : Springer Berlin Heidelberg, 1996, p. 1-17.

[121] Ramakrishnan SRIKANT, Quoc VU et Rakesh AGRAWAL. "Mining Association Rules with Item Constraints". In : *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*. KDD'97. Newport Beach, CA : AAAI Press, 1997, p. 67-73.

[122] Jiaojiao SUN et al. "Incremental Frequent Itemsets Mining With FCFP Tree". In : *IEEE Access* 7 (2019), p. 136511-136524.

[123] Laszlo SZATHMARY, Amedeo NAPOLI et Petko VALTCHEV. "Towards Rare Itemset Mining". In : *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*. T. 1. 2007, p. 305-312.

[124] Laszlo SZATHMARY et al. "Efficient Vertical Mining of Minimal Rare Itemsets". In : *Proceedings of The Ninth International Conference on Concept Lattices and Their Applications, Fuengirola (Málaga), Spain, October 11-14, 2012*. Sous la dir. de Laszlo SZATHMARY et Uta PRISS. T. 972. CEUR Workshop Proceedings. CEUR-WS.org, 2012, p. 269-280.

[125] Takahisa TODA et Takehide SOH. "Implementing Efficient All Solutions SAT Solvers". In : *ACM J. Exp. Algorithmics* 21 (nov. 2016). ISSN : 1084-6654.

[126] Gregory TSEITIN. "On the complexity of proofs in propositional logics". In : *Seminars in Mathematics*. T. 8. 1970, p. 466-483.

[127] Takeaki UNO. “Lcm ver. 3 : Collaboration of array, bitmap and prefix tree for frequent itemset mining”. In : *In Proc. of the ACM SIGKDD Open Source Data Mining Workshop on Frequent Pattern Mining Implementations*. 2005, p. 77-86.

[128] Takeaki UNO, Masashi KIYOMI et Hiroki ARIMURA. “LCM ver. 2 : Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets”. In : *FIMI ’04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*. Sous la dir. de Roberto J. Bayardo JR., Bart GOETHALS et Mohammed Javeed ZAKI. T. 126. CEUR Workshop Proceedings. CEUR-WS.org, 2004.

[129] Takeaki UNO et al. “LCM : An efficient algorithm for enumerating frequent closed item sets”. In : *In Proceedings of Workshop on Frequent itemset Mining Implementations (FIMI’03*. 2003.

[130] Bay VO, Tzung-Pei HONG et Bac LE. “DBV-Miner : A Dynamic Bit-Vector approach for fast mining frequent closed itemsets”. In : *Expert Syst. Appl.* 39.8 (2012), p. 7196-7206.

[131] Jianyong WANG et Jiawei HAN. “BIDE : Efficient Mining of Frequent Closed Sequences”. In : *ICDE*. IEEE Computer Society, 2004, p. 79-90.

[132] Ke WANG, Yabo XU et Jeffrey Xu YU. “Scalable sequential pattern mining for biological sequences”. In : *CIKM*. ACM, 2004, p. 178-187.

[133] Miao WANG, Xuequn SHANG et Zhanhuai LI. “Sequential Pattern Mining for Protein Function Prediction”. In : *ADMA*. T. 5139. Lecture Notes in Computer Science. Springer, 2008, p. 652-658.

[134] Yaling XUN et al. “Incremental frequent itemsets mining based on frequent pattern tree and multi-scale”. In : *Expert Syst. Appl.* 163 (2021), p. 113805.

[135] Xifeng YAN, Jiawei HAN et Ramin AFSHAR. “CloSpan : Mining Closed Sequential Patterns in Large Datasets”. In : *In SDM*. 2003, p. 166-177.

[136] Shengwei YI et al. “An effective algorithm for mining sequential generators”. In : *Procedia Engineering* 15 (2011), p. 3653-3657.

[137] Junfu YIN, Zhigang ZHENG et Longbing CAO. “USpan : an efficient algorithm for mining high utility sequential patterns”. In : *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, p. 660-668.

[138] Yinlei YU et al. “All-SAT Using Minimal Blocking Clauses”. In : *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. 2014, p. 86-91.

[139] Unil YUN. “Efficient mining of weighted interesting patterns with a strong weight and/or support affinity”. In : *Inf. Sci.* 177.17 (2007), p. 3477-3499.

[140] Unil YUN. “On pushing weight constraints deeply into frequent itemset mining”. In : *Intell. Data Anal.* 13.2 (2009), p. 359-383.

[141] Unil YUN et John J LEGGETT. “WSpan : Weighted Sequential pattern mining in large sequence databases”. In : *2006 3Rd international IEEE conference intelligent systems*. IEEE. 2006, p. 512-517.

[142] Unil YUN et John J. LEGGETT. “WFIM : Weighted Frequent Itemset Mining with a weight range and a minimum weight”. In : *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, Newport Beach, CA, USA, April 21-23, 2005*. Sous la dir. d’Hillol KARGUPTA et al. SIAM, 2005, p. 636-640.

[143] Unil YUN, Heungmo RYANG et Keun Ho RYU. “High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates”. In : *Expert Syst. Appl.* 41.8 (2014), p. 3861-3878.

[144] Mohammed J. ZAKI. “SPADE : An efficient algorithm for mining frequent sequences”. In : (2001), p. 31-60.

[145] Mohammed Javeed ZAKI. “Scalable Algorithms for Association Mining.” In : *IEEE Trans. Knowl. Data Eng.* 12.3 (2000), p. 372-390.

[146] Mohammed Javeed ZAKI et Ching-Jiu HSIAO. “CHARM : An Efficient Algorithm for Closed Itemset Mining”. In : *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*. Sous la dir. de Robert L. GROSSMAN et al. SIAM, 2002, p. 457-473.

[147] Lintao ZHANG et al. “Efficient conflict driven learning in a boolean satisfiability solver”. In : *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*. IEEE. 2001, p. 279-285.

[148] Souleymane ZIDA et al. “EFIM : A Highly Efficient Algorithm for High-Utility Itemset Mining”. In : *Advances in Artificial Intelligence and Soft Computing - 14th Mexican International Conference on Artificial Intelligence, MICAI 2015, Cuernavaca, Morelos, Mexico, October 25-31, 2015, Proceedings, Part I*. Sous la dir. de Grigori SIDOROV et Sofia N. GALICIA-HARO. T. 9413. Lecture Notes in Computer Science. Springer, 2015, p. 530-546.

Résumé

L'extraction de motifs fréquents est l'une des tâches fondamentales de la fouille de données. Elle consiste à découvrir des motifs intéressants pour l'utilisateur à partir des bases de données. Différents types des motifs peuvent être trouvés à partir de divers types de données, tels que les données transactionnelles, les séquences, les graphes et les arbres.

Cette thèse s'intéresse aux deux tâches bien connues de la fouille de motifs à savoir la fouille des itemsets fréquents et la fouille des motifs séquentiels.

La première contribution de cette thèse concerne l'utilisation des approches déclaratives de type SAT pour l'extraction des itemsets fréquents fermés. Ces approches sont connues pour être flexibles en permettant d'ajouter des contraintes pour extraire des motifs particuliers. L'un des problèmes majeurs de ces approches est le passage à l'échelle dû à la taille de l'encodage des grandes bases transactionnelles. Pour réduire la taille de ces encodages, nous avons proposé des représentations plus compactes. Plus précisément, nous montrons que notre approche peut être vue comme un problème de compression de matrices booléennes. Nos résultats montrent une réduction significative de la taille de l'encodage.

Notre seconde contribution consiste à exploiter les symétries dans le cadre de la fouille des motifs séquentiels. La recherche de ces symétries est effectuée en encodant la table des séquences sous forme d'un graphe orienté coloré. Les symétries sont alors énumérées en calculant les automorphismes du graphe. Finalement, nous proposons deux approches pour exploiter les symétries détectées. Dans la première, nous montrons comment nous pouvons améliorer l'extraction de motifs séquentiels en intégrant les symétries découvertes dans un algorithme de type Apriori. Dans la seconde approche nous démontrons comment ces symétries peuvent être utilisées en prétraitement. Cela est réalisé en modifiant la base de séquences originale.

Mots-clés: Motifs fréquents, itemsets, motifs séquentiels, symétries, satisfiabilité propositionnelle.

Abstract

Frequent pattern mining is one of the fundamental tasks of data mining. It consists in discovering interesting patterns for the user from the databases. Different types of patterns can be found from various types of data, such as transactional data, sequences, graphs and trees.

This thesis is concerned with the two well-known tasks of pattern mining, namely frequent itemsets mining and sequential pattern mining.

The first contribution of this thesis concerns the use of declarative approaches of the SAT type for mining frequent closed itemsets. These approaches are known to be flexible by allowing constraints to be added to mine particular patterns. One of the major problems of these approaches is the scalability due to the size of the encoding of large transactional databases. To reduce the size of these encodings, we have proposed more compact representations. More precisely, we show that our approach can be seen as a Boolean matrix compression problem. Our results show a significant reduction in encoding size.

Our second contribution consists in exploiting the symmetries within the framework of mining sequential patterns. The search of these symmetries is performed by encoding the sequence table in the form of a colored directed graph. The symmetries are then enumerated by calculating the automorphisms of the graph. Finally, we propose two approaches to exploit the detected symmetries. In the first, we show how we can improve the sequential pattern mining by integrating the symmetries discovered in an Apriori-like algorithm. In the second approach we demonstrate how these symmetries can be used in preprocessing. This is achieved by modifying the original sequences database.

Keywords: Frequent patterns, itemsets, sequential patterns, symmetries, propositional satisfiability.