وزارة التعليم العالي والبحث العلمي

BADJI MOKHTAR - ANNABA UNIVERSITY

UNIVERSITE BADJ MOKHTAR - ANNABA

جامعة باجي مختار - عنابة

Année : 2015

Faculté des Sciences de l'Ingéniorat

Département d'informatique

THÈSE

Présentée en vue de l'obtention du diplôme de Doctorat 3ème Cycle

Fouille de graphes: Algorithme combinatoire et Application

Filière : Informatique

Spécialité : Sciences et Technologies de l'Information et de la Communication (STIC).

Par :

Nour El Islem KARABADJI

DEVANT LE JURY

PRÉSIDENT:	Med Tarek Khadir	Professeur	Badji-Mokhtar-ANNABA.
Rapporteur :	Hassina Séridi	Professeur	Badji-Mokhtar-ANNABA.
Co-rapporteur :	Lhouari NOURINE	Professeur	LIMOS CLERMONT-FERRAND.
EXAMINATEURS:	Yamina Mohamed Ben Ali	Professeur	Badji-Mokhtar-ANNABA.
	Labiba souici-Meslati	Professeur	Badji-Mokhtar-ANNABA.
	Engelbert Mephu Nguifo	Professeur	LIMOS CLERMONT-FERRAND.

Remerciements

Ce manuscrit conclut quatre ans de travail, je tiens en ces quelques lignes à exprimer ma reconnaissance envers tous ceux qui de près ou de loin y ont contribué.

J'exprime en premier lieu ma gratitude au Pr Hassina Seridi, directrice de thèse, pour son encadrement, ses conseils, et pour sa disponibilité. L'idée de voler de ses propres ailes est un peu effrayante, mais j'ai l'impression d'avoir grandi, d'avoir acquis une certaine confiance grâce à vous. D'un point de vue relationnel, j'ai trouvé une relation cordiale, une écoute et ce que j'ai aimé par-dessus tout une franchise à toute épreuve. J'ai énormément appris à vos côtés. Merci pour tout.

Je souhaite néanmoins remercier plus particulièrement Pr Lhouari Nourine pour son aide précieuse et sa gentillesse. Merci à Ilyes Khelf et Nabiha Azzizi pour leur collaboration. Je souhaite adresser mes remerciements les plus sincères aux membres de jurys. Je remercie mes parents, et ma sœur pour leurs contributions, leurs patiences et leurs soutiens continuels sans fin. Je souhaite également remercier ma femme qui m'a toujours encouragée. J'ai trouvé à ses côtés, la force et le réconfort nécessaire pour mener ce travail à terme. Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amis, qui m'ont toujours soutenue et encouragée au cours de la réalisation de cette thèse.

Merci au lecteur qui par essence justifie la rédaction de ce document.

Je dédie cette thèse
à ma machine.

Résumé

Cette thèse traite d'un problème difficile qui est la découverte des motifs (modèles) intéressants à base de contraintes, en s'intéressant principalement à la découverte des sous-graphes connexes fréquents et partiellement à la découverte de modèles intéressants à base de contraintes. Les concepts fondamentaux du processus d'énumération, ainsi que l'espace de recherche combinatoire, ont été étudiés et présentés.

Pour la découverte des sous-graphes connexes fréquents, les différentes difficultés liées au problème concernant la représentation des graphes, les opérations et l'espace de recherche ont été explorés ainsi que la majorité des techniques de découverte. Les contributions de la thèse concernent une classe particulière des graphes, qui ont un étiquetage non redondant d'arêtes, et se matérialisent par la proposition de deux approches de découverte de sous graphes connexes fréquents fermés. Le principal apport était la restriction de l'espace de recherche en minimisant le nombre des sous-graphes non intéressants (c.-à-d. fréquents non fermés) visités avant de récupérer tous les sous-graphes fréquents fermés. En plus de cette restriction de l'espace de recherche, chacune des deux approches adopte sa propre représentation des graphes. L'une permet de réduire le problème d'isomorphisme de graphes et de sous graphes, à une relation d'équivalence et d'inclusion d'ensemble, en s'appuyant sur un encodage des graphes à étiquetage d'arêtes non redondant en des ensembles d'items. Quant à la seconde approche, elle permet de résoudre efficacement le test d'isomorphisme de graphes et de sous graphes pour le moyen des cas, et même en un temps linéaire pour le meilleur des cas, et ce en représentant les graphes comme des tableaux d'adjacences.

Pour la découverte de modèles intéressants à base de contraintes, une approche qui permet l'amélioration de la qualité d'un modèle (arbre de décision dans notre cas) lors de la phase d'augmentation a été proposée. La contribution primaire porte essentiellement sur la résolution des problèmes de construction d'un arbre de décision, liés à la complexité du modèle et au sur-apprentissage ; par recherche de la combinaison optimale composée d'un sous-ensemble d'attributs et un sous-ensemble d'exemples d'apprentissage sans avoir recours à la phase d'élagage.

Mots-clés: Data mining, Problèmes combinatoires, L'énumération, Graphes fréquents, Modèles intéressants.

Abstract

This thesis deals with a challenging problem that is Constraint-based mining, which refers the discovery of interesting patterns (models) based on constraints. At this thesis, we are mainly interested to frequent connected subgraphs mining problem and partially to the interesting models based constraints discovery. The basic concepts of the listing process, as well as combinatorial search space have been studied and presented.

According to the frequent connected subgraphs discovery problem, one, we present various difficulties about the problem : graphs representation, the different subtasks over them, and the search space complexity. Next, an interesting number of frequent subgraphs discovery algorithms are presented. Contributions presented toward subgraphs discovery problem is intended for unique edges labels graphs. We propose two approaches to discovery frequent connected subgraphs closed. The main contribution of these two approaches is to restrict the search space by minimizing the number of non-interesting subgraphs (i.e. frequent non-closed). However, each approach uses a different graphs representation. One serves to reduce the graph and subgraphs isomorphism problems into equivalence and inclusion sets relations, based on an encoding of graphs to an Edge and Converse Edges items sets. While the second approach serves to effectively solve the graph and subgraph isomorphism problem for the average cases, and even in linear time for the best case.

For the discovery of interesting patterns models based on constraints, a new approach that avoids the over-fitting and complexity problems suffered in the construction of decision trees (the models) is proposed. In the present work, a combination of attributes selection and data sampling is used to overcome the construction problems.

Keywords: Data mining, Combinatorial problems, Enumeration, Frequent graphs, Interesting models.

الملخص

إن هذه الأطروحة تتناول مشكلة التنقيب في البيانات على أساس مجموعة من القيود، وهي عملية بحث واسكتشاف لأنماط (نماذج) مثيرة للاهتمام على أساس القيود. بعد طرح المفهيم الرئسية لعملية العدّ، فضلا عن فضاء البحث التوافقي. إن هذه الأطروحة، تهتم أساسا بستكشاف المخططات على اساس تكرر ظهورها بالنسبة الى قاعدة من المعطيات، وتهتم جزئيا لبناء شجرة القرار على اساس القيود.

وفقا للمشكل الأسسي، فإننا نستعرض أولا مختلف الصعوبات التي تعقد عملية البحث: تمثيل المخططات، ومختلف العمليات (تشاكل المخططات...)، وصعوبة فضاء البحث. وبعد ذلك، يتم عرض عدد هام من خوارزميات اكتشاف المخططات. بالنسبة لهذا الموضوع لقد تم إقتراح طريقتين جديدتين لستكشاف المخططات المغلقة على اساس تكرر ظهورها بالنسبة إلي المخططات المتميزين باضلع ذات العلامات فريدة. الإضافة الرئيسية لهاتين المقاربتين هو فرض قيود على فضاء البحث عن طريق التقليل من عدد التوافقات غير المثيرة للاهتمام (أي متكررة غير مغلقة). بيد أن، كل تقنية تستعمل تمثيل مختلف بالنسبة للمخططات، واحدة تعمل على تحويل مشكل تشاكل المخططات إلى علاقات التكافؤ وإدراج مجموعات، أما بالنسبة للتمثيل الثاني فتعمل على إيجاد حل لمشكل تشاكل المخططات في زمن معقول للحالات متوسطة الصعوبة، وحتى في زمن خطي في أفضل الحالات.

بالنسبة المشكلة الثانية، لقد تم إقتراح مقاربة جديدة تهدف لتجنب التعقيدات و المشاكل التي تعاني منها مرحلة إنشاء أشجار القرار. حيث يتم استخدام عملية مزج لاختيار المؤشرات والبيانات للتغلب على مشاكل الإنشاء.

كلمات البحث: التنقيب في البيانات، العدّ، مشاكل التوافقات، إستكشاف المخططات على اساس تكرر ظهورها، إستكشاف النماذج مثيرة للاهتمام على أساس القيود.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

*I*ntroduction générale

Cette thèse porte essentiellement sur les modèles de représentation des données pour la fouille de données. Nous avons investigué aussi bien le domaine de la représentation en graphe et la fouille des graphes ainsi que le domaine de la représentation en arbres de décision pour leur utilisation dans la fouille de données.
Étant donné que la modification de titre est une tâche administrative quasi impossible, nous avons gardé le titre initial et nous avons extrapolé notre étude aux deux modèles de représentation précédemment cités.

Depuis son apparition, l'ordinateur avait principalement permis de décharger l'être humain de la pénible tâche de chercher des solutions optimales parmi l'ensemble de toutes les configurations possibles. Pour résoudre cette tâche, une solution satisfaisante (ou un ensemble de solutions) est atteinte généralement en explorant toutes les configurations alternatives possibles sur un espace de recherche. Ce processus de recherche permet de récupérer soit une, ou plusieurs solutions, et peut même ne pas renvoyer de solutions, et ce en corrélation avec les critères de satisfactions exigés.

À titre d'exemple, soit un ensemble de tâches avec des temps d'exécutions connus, nous devons affecter les tâches à des groupes d'employés. Chaque tâche peut être confiée à un seul employé ou à un groupe d'employés c.-à-d. un ou plusieurs employés peuvent être affectés à une même tâche. Chaque employé peut travailler sur plusieurs tâches, mais en revanche, il ne peut pas exécuter plusieurs tâches simultanément. L'objectif de cette affectation est d'exécuter l'ensemble des tâches aussi rapidement que possible. Afin de résoudre efficacement ce problème d'ordonnancement, l'idée est de minimiser le nombre d'employés affectés à plusieurs tâches différentes. Cela est dû au fait que le temps d'exécution de l'ensemble des tâches est alors égal au temps de travail de l'employé le plus occupé, étant donné qu'un employé ne peut réaliser qu'une tâche à la fois. Par consé-

quent, la meilleure affectation est celle qui permet la restriction d'un nombre important (maximal) d'affectations d'un même employé à plusieurs tâches. Cependant, atteindre la meilleure affectation revient à la chercher parmi l'ensemble de toutes les affectations possibles.

L'avantage de ce processus de recherche de solutions au sein d'un ensemble d'alternatives est d'avoir la certitude de récupérer toutes les solutions existantes, ainsi que l'affirmation de l'inexistence d'aucune solution pour le cas où aucune solution ne satisfait les critères appliqués pour la recherche. Cependant, cette recherche nécessite l'exploration de toutes les configurations possibles ce qui incite à tester si toutes ces configurations satisfont bien ou non les critères de recherche. Généralement, ces configurations sont issues de la combinaison d'un ensemble d'éléments, et la génération de toutes les combinaisons possibles est d'ordre exponentiel. Ce nombre de combinaisons nécessite un temps de calcul énorme, et malgré la puissance croissante des machines, l'exploration de toutes les combinaisons est toujours un cas délicat et même intraitable en pratique.

Prenons comme exemple un groupe de n employés, le nombre de combinaisons d'employés possibles est 2^n. Pour des données de petite taille, il est possible d'explorer toutes les combinaisons dans le pire des cas, mais cela est pratiquement impossible pour des données de grande taille. Les problèmes utilisent l'exploration des espaces de configurations (combinaisons, motifs, modèles, etc.), sont inscrits comme des disciplines de découverte de motifs ou modèles intéressants à base de contraintes. Ce dernier domaine de recherche est plutôt connu aussi comme la résolution de problèmes par recherche.

Ce manuscrit de thèse traite deux problèmes qui souffrent d'explosion combinatoire et impliquent l'exploration d'un espace de configurations afin de chercher des solutions exactes (ou une seule solution exacte). Principalement, nous traitons la découverte des sous-graphes connexes fréquents. Ce dernier problème consiste à chercher un ensemble de sous-graphes connexes ayant un certain nombre d'occurrences par rapport à une base de graphes, parmi tous les sous-graphes connexes possibles. Partiellement, nous nous intéressons dans cette thèse, à la construction d'arbres de décisions (AD) qui va être représentée comme étant un problème de fouille de modèles intéressants à base de contraintes. Pour ce dernier problème, nous cherchons à récupérer un arbre de décision robuste parmi un ensemble d'arbres de décisions qui ont été construits en variant un couple d'exemples d'apprentissages et des ensembles d'attributs.

1.1 La découverte des sous-graphes connexes fréquents

Récemment, nous assistons à diverses possibilités d'échange et de stockage de plus en plus d'information, ainsi que de nombreuses applications qui sont représentées naturellement comme des graphes : Réseaux sociaux et routiers, au niveau de l'analyse des réseaux sociaux, un graphe est composé d'un ensemble d'individus (profils) liés par des arêtes qui expriment les relations entre ces personnes. Alors que pour les réseaux routiers, chaque noeud (sommet) représente une ville ou un endroit et les arêtes entre les sommets représentent des routes directes entre ces villes. En Bio-informatique, un graphe représente un réseau d'interaction protéine-protéine qui apparaîtrait lorsque deux ou plusieurs protéines (sommets) se lient entre elles, où un lien (arête) entre deux protéines signifie que les protéines participent à une même fonction biologique particulière. En chimio-informatique, les graphes encodent parfaitement l'information structurelle et cyclique des molécules. Pour les Combinaisons, la relation d'ordre entre des dispositions d'objets sans tenir compte de l'ordre de placement de ces objets, qui prend souvent la forme d'un treillis, peut être représentée comme un graphe acyclique dont les noeuds sont les dispositions d'objets, et les arêtes correspondent à des relations de couvertures (des comparabilités immédiates) entre les dispositions (p. ex. diagramme de hasse), etc.

Ces dernières années, une importante tendance envers la fouille des graphes a vu le jour. Cette tendance s'est résumée principalement par le développement de techniques pour analyser, explorer et extraire des connaissances à partir des graphes. En contrepartie, il est plus difficile de manipuler ces structures qui parfois sont même intraitables en pratique due à leurs complexités. La difficulté liée à la manipulation de ces structures peut se résumer à faire face à de nombreux problèmes : d'abord le test d'isomorphisme de graphes et de sous-graphes, ainsi que la génération et l'exploration des graphes sans redondances et enfin la représentation en mémoire de ces derniers.

L'un des domaines les plus populaires de la fouille des graphes est la découverte des sous-graphes connexes fréquents. Comme nous l'avons déjà noté, cette discipline s'inscrit comme une spécialisation d'un problème plus général qui a été largement étudié dans la littérature "La découverte des motifs fréquents". La découverte des sous-graphes connexes fréquents peut être définie comme suit, soit une base de données $\mathcal{D}$, un seuil de fréquence δ, alors l'objectif est de récupérer tous les sous-graphes connexes qui ont une occurrence d'au moins δ fois au niveau de la base $\mathcal{D}$.

Beaucoup d'algorithmes ont été développés pour l'énumération des sous-graphes connexes fréquents. Cependant, ces algorithmes souffrent de l'isomorphisme de graphes lors de la phase de génération des candidats et de l'isomorphisme de sous-graphes pour tester la fréquence. Ces deux derniers problèmes sont connus difficiles pour le cas général. Pour l'isomorphisme de graphes, aucun algorithme polynomial n'est connu, alors que même la complexité du problème n'est pas encore connue à ce jour (c.-à-d. P ou $NP-complet$). Pour l'isomorphisme des sous-graphes, le problème est encore beaucoup plus complexe et connu $NP-complet$. En plus des problèmes d'isomorphismes, ces algorithmes souffrent aussi de la présence d'un nombre important de sous-graphes fréquents, ce qui rend l'énumération de l'intégralité des sous-graphes ainsi que l'analyse quasi impossible en raison du fait que tous les sous-graphes (généralisations) d'un sous-graphe fréquent donné X sont également fréquents. Ce phénomène est dû à la propriété de fermeture descendante. Pour appréhender ce grand nombre de sous-graphes, la fouille des sous-graphes fréquents fermés $\mathcal{C}$ et ceux maximaux $\mathcal{M}$ a été proposée. La principale motivation est que l'ensemble des sous-graphes fréquents fermés et maximaux est largement plus petit que celui de l'ensemble complet des sous-graphes fréquents $\mathcal{F}$.

Cette dernière tendance vers l'énumération des sous-graphes connexes maximaux et fermés est aussi motivée par rapport au fait que les sous-graphes fréquents $\mathcal{F}$ peuvent être régénérés à partir des sous-graphes maximaux $\mathcal{M}$ et ceux des fermés $\mathcal{C}$.Encore, le nombre des sous-graphes maximaux et fermés reste nettement inférieur au nombre de tous les fréquents $\mathcal{F}$. Par conséquent, un nombre important d'opérations de test de fréquence et de génération de candidats peuvent être évitées. Cependant, pour investir cette restriction il faut réduire le nombre des sous-graphes fréquents non intéressants (ic.-à-d. fréquents non fermés et/ou non maximaux) à visiter avant d'atteindre les fermés et/ou les maximaux. Malheureusement, contrairement à des cas moins complexes comme les itemsets et les arbres, où les motifs maximaux et motifs fermés sont récupérés en explorant uniquement un ensemble de motifs d'ordre équivalant à celui des maximaux $\mathcal{M}$ et celui des fermés $\mathcal{C}$ respectivement, l'espace de recherche dans le cas des graphes est beaucoup plus complexe et limite les manoeuvres de restriction des sous-graphes considérés par un processus de recherche. Cependant, l'accessibilité forte de cet espace de recherche permet d'investir l'existence d'une chaine CH de taille t de sous-graphes comparables entre chacun des couples des sous-graphes $(c_1, c_2$, t.q $c_1 \subset c_2)$ fermés successifs tels que pour chacun des sous-graphes g_i d'une chaine entre deux sous-graphes fermés CH (c.-à-d. $\forall\ g_i, \in CH$), $E(c_1) \subseteq E(g_i)$, $E(g_i) \subseteq E(c_2)$, alors il existe un autre graphe $g_j \in CH,\ t.q\ \ E(g_i) \subseteq E(g_j)$ et $|E(g_j| = |E(g_i)| + 1$. Cette dernière propriété ouvre la possibilité de sauter d'un graphe fermé vers un fermé successeur du premier sans parcourir la séquence des graphes entre

les deux, en calculant un opérateur de fermeture, si ce dernier est bien existant.

La principale contribution de cette thèse est d'avoir proposé deux approches de découverte des sous-graphes connexes fréquents fermés à étiquetage d'arêtes non redondant. Ces deux approches investissent la forte accessibilité de l'espace de recherche pour réduire le nombre des sous-graphes non intéressants en essayant de sauter d'un sous-graphe fermé vers au moins l'un de ses fermés successeurs. Cependant, le fait qu'un graphe fermé admet plusieurs successeurs qui peuvent être générés, alors il n'existe pas un opérateur de fermeture pour lequel nous pouvons atteindre directement l'ensemble des successeurs fermés. Par conséquent, pour récupérer l'ensemble des successeurs fermés d'un sous-graphe donné, les deux approches proposées s'exécutent différemment. L'une utilise un codage pour réduire le problème en un calcul de la fermeture d'un ensemble d'items par l'intersection d'une collection d'ensembles d'items. Alors que pour l'autre, cet ensemble des fermés successeurs est cherché directement en appliquant un algorithme de recherche en profondeur en lui associant quelques optimisations afin de réduire le nombre des candidats.

Autres que cette tentative de réduction de l'espace de recherche, nos contributions ont porté aussi sur la réduction de la complexité des coûts de génération des candidats et de tests de leurs fréquences. Par rapport à la génération des candidats, nous devons résoudre trois opérations : l'augmentation (c.-à-d. à partir d'un sous-graphe déjà traité, générer un nouveau en lui ajoutant une nouvelle arête), le test de connexité (c.-à-d. tester que le nouveau sous-graphe généré est bien connexe), et la non-redondance (c.-à-d. que ce nouveau n'a pas été généré auparavant). La dernière opération est la plus coûteuse, et consiste à tester l'isomorphisme des graphes par rapport à tous les sous-graphes qui ont été déjà générés. Nous allons illustrer l'impact de représentation des graphes vis-à-vis la résolution des trois opérations, et présenter des expérimentations afin de donner une image pertinente de cet impact. Nous allons adopter deux représentations différentes, une ensembliste, et l'autre comme tableaux d'adjacences. Pour l'approche qui utilise le codage vers des ensembles d'items, nous avons opté pour une représentation ensembliste, et pour l'autre nous avons opté pour une représentation en tableaux d'adjacences. La représentation ensembliste permet de résoudre efficacement les opérations d'augmentation et de test de connexité, et elle permet de faciliter la transition vers le codage d'ensembles d'items, où la complexité de test d'isomorphisme de graphes et de sous-graphes est réduite à un test d'équivalence et d'inclusion d'ensembles d'items, ce qui réduit considérablement la complexité des tests de fréquences pour le pire des cas. Par ailleurs, l'adoption d'une représentation par tableaux d'adjacences permet de réduire la complexité du test de fréquence par rapport au meilleur et moyen des cas, où la résolution du test est d'ordre linéaire pour

le meilleur des cas. Enfin par rapport à cet axe, les expérimentations qui ont été conduites à l'égard des deux approches proposées sont présentées.

1.2 La découverte de modèles à base de contraintes : Application pour la construction d'un AD

La fouille de données est un processus complexe, devenu central aujourd'hui soit au niveau des systèmes d'aide à la décision que dans le développement des systèmes de décisions complètement automatisés. Ce processus peut être défini simplement comme étant une discipline qui vise l'exploitation de grandes quantités de données collectées dans divers domaines d'applications. Depuis longtemps, de nombreux travaux ont été élaborés pour découvrir l'information contenue dans une base de données. À ce titre, beaucoup de disciplines ayant connu beaucoup de succès : l'extraction de motifs intéressants, sélection d'attributs, classification, prédiction, etc.

Parmi les techniques de la fouille de données, le classement a été distingué comme étant l'une des opérations les plus populaires. Cette technique de classement vise la construction d'un modèle de classement (c.-à-d. un classificateur) à partir d'un ensemble d'exemples "bien choisis" pour lesquels les classes sont connues, afin que le modèle soit capable de généraliser et d'être prêt pour classer de nouvelles instances (c.-à-d. un apprentissage supervisé). Les exemples utilisés pour la construction ont été classés généralement par un expert ou à la base des résultats expérimentaux (p. ex. la météo, etc.). Autrement dit, ce processus de classement vise la découverte des caractéristiques remarquables d'un ensemble de données qui permettent de distinguer les classes des nouvelles instances qui partagent des caractéristiques similaires.

Techniquement, le classement consiste en la définition d'une fonction qui permet d'affecter chaque entrée des données (les exemples d'apprentissages) dans la bonne classe parmi plusieurs classes prédéfinies. Cette fonction est paramétrée en fonction des caractéristiques des exemples d'apprentissages. Cependant, plusieurs fonctions peuvent être estimées pour un même échantillon de données, alors que la fonction espérée est celle qui sera capable de bien classer de nouveaux exemples qui n'ont pas été considérés lors de la phase d'apprentissage. La résolution de problème revient alors à trouver la fonction $h(x)$ parmi l'ensemble de toutes les fonctions possibles. La fonction espérée est donc celle qui minimise l'erreur par rapport à un échantillon d'exemples de test indépendant. À partir de cette dernière définition, un processus d'apprentissage supervisé pour la construction d'un modèle de classement est naturellement formulé sous la forme d'un problème de

fouille de modèle intéressant à base de contraintes. D'autre part, le choix de l'échantillon d'apprentissage, ainsi que la sélection d'attributs sont aussi deux problèmes d'une nature combinatoire associés au processus de construction d'un classificateur. Pour la phase d'apprentissage, la variation d'exemples d'apprentissage et l'ensemble d'attributs permettent la génération d'un modèle différent pour chaque instance.

Les arbres de décisions sont l'une des techniques les plus populaires pour le classement. Ils ont été largement utilisés dans les applications d'ingénieries, car les arbres de décisions permettent aux utilisateurs de comprendre facilement le comportement des modèles construits. La construction d'un arbre de décision en suivant les modèles heuristiques existants dans la littérature (p. ex. J48, SimpleCart, ID3, BFTree, etc.), suit deux étapes principales : une phase d'agrandissement de l'arbre (c.-à-d. le faire pousser) et une phase d'élagage pour tailler les règles issues de données incertaines et/ou erronées. La phase d'agrandissement consiste en l'opération de choix d'un attribut qui permet la meilleure séparation d'une population de façon à avoir deux sous-populations, en utilisant un critère de séparation distinct (c.-à-d. Gini, Gain d'information, χ^2, etc.). Cette opération est réitérée sur chacune des sous-populations que nous appelons noeuds, jusqu'à ce qu'aucune séparation ne soit possible. Pour cette phase, l'utilisation d'exemples d'apprentissages différents permet la génération d'arbre de décision différent.

Un arbre de décision issue d'une phase d'agrandissement souffre généralement d'un sur apprentissage, comme pour tout classificateur, la construction d'un arbre de décision s'appuie sur les données d'entrées et l'objectif consiste à utiliser une bonne partie de ces données pour construire l'arbre de décision qui minimise l'erreur de précision de la classification par rapport à un ensemble indépendant d'exemples de tests. Cependant, l'objectif ne sera jamais atteint sans l'élagage de l'arbre afin de tailler les règles (c.-à-d. les sous-arbres) issues de données incertaines ou erronées. Autrement dit, la construction d'un arbre de décision peut être définie comme la recherche du sous-arbre de décision le plus performant par rapport à l'échantillon de test parmi tous les sous-arbres possibles de l'arbre issu de la phase d'agrandissement.

Dans la littérature, il existe de nombreuses variantes de techniques d'élagage, et les expérimentations [Esposito *et al.*1997], [Bradford *et al.*1998], [Mingers1989] ont illustré qu'aucune de ces méthodes ne domine les autres. Ces expérimentations ont montré que les performances des méthodes d'élagages dépendent fortement de la taille, du bruit, de la qualité des attributs et du type des données de la base de données manipulée. Cette multiplicité et diversité des modèles implique une étude approfondie des données ainsi

que les techniques et les critères appliqués pour chacune des méthodes d'élagages, pour enfin pouvoir choisir celle qui sera la plus appropriée pour l'élagage de l'arbre de décision généré lors de la phase d'augmentation. Malgré leurs nombreux avantages, les méthodes d'élagages souffrent principalement des problèmes de sur élagage des arbres de décisions, ainsi que le sous-élagage. Cela est dû au fait que les méthodes proposées ne prennent en charge qu'une famille restreinte de tous les sous-arbres de décisions possibles (c.-à-d. heuristiques).

Autre que les techniques utilisés en pratique, théoriquement, la construction d'un arbre de décision optimal à base de certaines contraintes, peut être définie comme suit : 1) chercher l'arbre de décision qui maximise les performances par rapport à un ensemble indépendant d'exemples de test parmi l'ensemble des arbres de décisions construits à partir de tous les sous-ensembles d'exemples de l'échantillon d'apprentissage. 2) En poussant un arbre de décision T_{max} en suivant un modèle donné (p. ex. J48, Cart, ou RandomTree, etc.), et ensuite, chercher l'arbre de décision qui maximise les performances par rapport à un ensemble d'exemples indépendants de tests parmi tous les sous-arbres de décisions de T_{max}. Ces deux solutions sont performantes, mais très complexes.

Cependant, en éliminant un sous-ensemble d'attributs non intéressants (c.-à-d. sans influence, peu influents, redondants), et ainsi un sous-ensemble d'exemples non intéressants (c.-à-d. contre-exemples, exemples mal classés, etc.), nous augmentons les chances de construction des arbres de décisions performants sans avoir besoin d'avoir recours aux méthodes d'élagages. L'idée consiste à chercher une combinaison d'un sous-ensemble d'attributs et un sous-ensemble d'exemples d'apprentissages par rapport à toutes les combinaisons possibles. La combinaison à extraire est celle qui maximise les performances par rapport à un ensemble indépendant d'exemples de test.

Dans ce manuscrit, nous proposons aussi une nouvelle approche pour la construction d'un arbre de décision par recherche. Motivé par le fait que le problème de construction d'un arbre de décision n'est qu'une découverte du modèle le plus performant à base de contraintes fixées par un utilisateur, et aussi par les avantages de la sélection d'attributs et d'avoir le bon échantillon d'apprentissage. Cette approche tente la construction d'un arbre de décision performant sans avoir recours à une phase d'élagage. Autrement dit, cette approche est une technique qui permet la construction d'un arbre de décision non élagué ayant des performances optimales par rapport à un ensemble d'exemples de test en accord avec les contraintes fixées par un utilisateur. L'idée derrière cette proposition est d'investir des solutions de traitement de données incertaines. Alors nous avons réduit le

problème à une recherche de la combinaison d'un sous-ensemble d'attributs et d'exemples qui permet la construction de l'arbre de décision ayant la meilleure performance.

1.3 Organisation du mémoire

La première partie (chapitre 2, et 3) de cette thèse s'intéresse d'une part, à introduire le problème de la découverte des motifs et des modèles intéressants à base de contraintes. D'autre part, à présenter principalement un état de l'art de la découverte des sous-graphes connexes fréquents et une étude poussée sur la découverte des sous-graphes connexes fréquents maximaux et fermés. Cette première partie s'intéresse aussi partiellement à la construction des arbres de décisions par recherche.

Le chapitre 2 de ce mémoire s'intéresse aux définitions, les propriétés, et les notations fondamentales des structures que nous allons utiliser tout au long de cette thèse : la découverte des motifs et modèles intéressants, la théorie des graphes et les problèmes combinatoires. Les objectifs de ce chapitre sont : (i) la présentation en détail du problème de découverte des motifs et modèles intéressants afin de sensibiliser le lecteur à l'importance de cette discipline qu'est une généralisation de nombreux problèmes combinatoires. (ii) expliquer les notions de l'espace de recherche de motifs (modèles), ainsi que les relations entre ces motifs (modèles). (iii) illustrer les notions de la théorie des graphes afin d'illustrer les difficultés liées à la manipulation de ce type de structures complexes.

Dans le chapitre 3, nous présentons l'état de l'art de la découverte des motifs fréquents. Par rapport à la découverte des motifs fréquents, nous illustrons trois de ses spécialisations les plus populaires. Pour ces trois spécialisations, les motifs sont : (i) les ensembles d'items, (ii) les sous-arbres connexes, (iii) les sous-graphes connexes. L'illustration de la découverte de ces trois types de motifs ayant des niveaux de complexité croissante a pour but de sensibiliser le lecteur à la difficulté du problème de découverte des sous-graphes connexes. La complexité des graphes affecte les opérations élémentaires, alors, la génération des candidats est beaucoup plus complexe à cause du grand nombre de possibilités d'extensions. Le test de fréquence est un test d'isomorphisme de sous-graphes qui est un problème $NP - complet$. Le nombre exponentiel de motifs, où même pour les cas des types non complexes, rend le problème intraitable. Afin de contourner ce problème, la découverte des motifs fréquents fermés et maximaux a été proposée. Enfin, ce chapitre présente en détail les différentes approches qui ont été proposées dans la littérature pour la découverte des sous-graphes connexes fréquents fermés et maximaux.

La deuxième partie de cette thèse (chapitre 4, et 5) traite nos contributions avec rapport à la découverte des sous-graphes connexes fermés fréquents à étiquetage non redondants et la construction d'un arbre de décision en recherchant la combinaison optimale d'un sous-ensemble d'exemples d'apprentissages et d'attributs, respectivement.

Nous traitons dans le chapitre 4 le problème de la découverte des sous-graphes fermés connexes fréquents à étiquetage d'arêtes non redondant. Nous présentons deux nouvelles approches pour la découverte de ce type de graphes. Nous proposons principalement l'investissement de la forte accessibilité de l'espace de recherche afin de réduire considérablement les sous-graphes connexes fréquents non fermés à visiter. Deux représentations mémoires des graphes sont utilisées pour la première approche notée **ECERCSgl** (**E***dges and* **C***onverses* **E***dges* **R***epresentation* **C***losed* **S***ub*$-$**g***raphs* **l***ist*), une représentation ensembliste, et pour la deuxième notée **ADARCSgl** (**AD***jacency* **A***rray* **R***epresen*$-$*tation* **C***losed* **S***ub* $-$ **g***raphs* **l***ist*), une représentation comme tableaux d'adjacences. Ces deux représentations permettent la réduction du problème de test d'isomorphisme des sous-graphes à : (i) un test de relation de sous-ensemble, en encodant les graphes comme des ensembles d'items. (2) une comparaison de tableaux d'adjacences qui peut être peu coûteuse pour le cas moyen et même en un temps linéaire dans le meilleur des cas. Enfin, nous présentons les résultats obtenus sur diverses configurations de base de graphes synthétiques connexes de ces deux approches.

Dans le chapitre 5, nous proposons une approche de construction d'arbres de décisions sans avoir recours à une phase d'élagage des arbres générés. Cette approche est notée $\mathcal{IUDT}$ (Improved Unpruned Decision Tree) et cherche à récupérer la combinaison de l'ensemble d'exemples d'apprentissages X et d'attributs A optimale. À ce niveau, nous exprimons par optimal, la combinaison qui maximise la précision de classification de l'arbre de décision construit à base de cette combinaison. L'arbre de décision qui maximise la performance est construit en divisant les exemples X par les attributs A. Dans ce même chapitre, nous présentons les résultats expérimentaux de $\mathcal{IUDT}$ en utilisant 10 bases de données de différents domaines, issue d'UCI [Blake and Merz1998]. Le dernier chapitre conclut l'ensemble des travaux présentés et présente nos perspectives.

Première partie

Concepts de base et travaux connexes

Chapitre 2

*C*oncepts fondamentaux

Sommaire

Ce chapitre introduit les concepts préliminaires, les définitions et les notions fondamentales du domaine de recherche traité par cette thèse. Principalement, nous présentons la fouille de données à base de contraintes, la découverte des motifs intéressants, la représentation du problème, ses problèmes dominants, et quelques notions sur la théorie de graphes.

2.1 Les problèmes combinatoires

Comme nous l'avons déjà mentionné, l'optimisation combinatoire est une discipline qui se positionne à l'intersection de l'informatique et les mathématiques et consiste à trouver la meilleure solution (ou un ensemble de meilleures solutions) par rapport à une fonction objective, dans un espace de recherche discret de solutions. Par définition, un problème d'optimisation combinatoire est :

Definition 2.1.1 *Un problème d'optimisation combinatoire consiste à déterminer le minimum (maximum) s^* d'une application f, sur l'ensemble des images finis de f :*

$$f(s^*) =_{s \in S} min\{f(s)\}$$

Beaucoup de problèmes de la vie réelle sont naturellement présentés comme des problèmes d'optimisations combinatoires [Marmion2011] tels que le transport, la gestion, et le partage des ressources, etc. À titre d'exemple, soit un distributeur automatique de boissons, le nombre de boissons à choisir est de 12 types différents. Le nombre de combinaisons de deux types de boissons possibles est de $\frac{n!}{(n-r)!}$ alors que $n = 12$ et $r = 2$ ce qui fait 66 possibilités. Clairement, juste le fait de générer ces combinaisons peut être intraitable pour une taille de données plus importantes. Alors que le problème peut être encore plus compliqué en lui associant une fonction objective qui nous permet de choisir une combinaison particulière.

Les méthodes développées pour résoudre les problèmes d'optimisation sont principalement catégorisées en deux classes, des méthodes exactes, et d'autres approchées. Les méthodes exactes explorent implicitement toutes les solutions de l'espace de recherche ce qui garantit la complétude de la résolution du problème, mais ces méthodes souffrent de la taille du problème qui est souvent importante, ce qui nécessite un temps de calcul exponentiel par rapport à la taille du problème traité. Contrairement à la première classe, les méthodes approchées (c.-à-d. heuristiques) sacrifient l'exploration complète de l'espace de recherche au profit de la rapidité ce qui permet de traiter des problèmes d'optimisation de grande taille si l'optimalité n'est pas la préoccupation principale.

Ayant un ensemble de solutions alternatives, et le fait de chercher la solution optimale parmi elles en se basant sur un certain nombre de contraintes, le domaine de recherche est plutôt connu sous le nom de la fouille de données (solutions) à base de contraintes (Constraint-Based Mining) [Nijssen2010a]. Autrement dit, la fouille de données à base de contraintes (Constraint-Based Mining) est un domaine de recherche qui s'intéresse au développement d'algorithmes qui permettent la découverte au sein d'un espace de motifs ou de modèles, d'un sous-ensemble d'éléments (c.-à-d. motifs ou modèles) réduit pour lequel les éléments sont distingués comme intéressants par rapport à certaines contraintes.

Comme il a été déjà mentionné, nous nous intéressons dans cette thèse à deux problèmes de nature combinatoires, principalement à la découverte des sous-graphes connexes fréquents qui est une spécialisation de la fouille des motifs intéressants à base de contraintes. Partiellement, nous nous intéressons aussi à la construction d'un arbre de décision sous contraintes qui est une autre spécialisation la fouille de modèles à base de contraintes.
Par rapport à la découverte des sous-graphes connexes fréquents, la résolution du problème consiste à parcourir l'espace de recherche de tous les sous-graphes connexes, pour enfin récupérer que les fréquents parmi eux. Pour ce dernier cas, le problème combi-

natoire est modélisé comme un espace de recherche composé de tous les sous-graphes connexes possibles et un prédicat qui permet de valider ou non un sous-graphe connexe testé comme étant fréquent ou non par rapport à un seuil défini par un utilisateur. De même, la construction d'un arbre de décision sous contraintes consiste à chercher le sous-arbre de décision le plus robuste parmi tous les arbres de décisions possibles pour un certain jeu de données.

2.2 La fouille des motifs intéressants à base de contraintes

Connue beaucoup plus par l'une de ces spécialisations (la découverte des motifs fréquents), la fouille des motifs intéressants à base de contraintes vise à résoudre un problème d'optimisation en utilisant des méthodes exactes. La résolution du problème consiste à parcourir l'espace de recherche de tous les motifs ou modèles afin de récupérer que ceux qui satisfont les contraintes parmi eux. Beaucoup plus connu pour le cas des motifs, cette discipline a été formulée par Mannila et Toivonen [Mannila and Toivonen1997], et a été largement étudiée dans la littérature [Mannila and Toivonen1997], [Ganter and Reuter1991], [Gunopulos *et al.*2003], [Chi *et al.*2005].

Definition 2.2.1 *(Découverte des motifs intéressants). Soit une base de données $\mathcal{D}$, un langage $\mathcal{L}$ qui exprime des motifs ou des modèles quelconques, et un prédicat de sélection $\mathcal{P} : \mathcal{L} \rightarrow \{vrai, faux\}$, l'objectif est de trouver une théorie de $\mathcal{D}$ selon $\mathcal{L}$ et $\mathcal{P}$,*

$$Th(\mathcal{D}, \mathcal{L}, \mathcal{P}) = \{\varphi \in \mathcal{L} | \mathcal{P}(\mathcal{D}, \varphi) = vrai\}.$$

En général, un processus de découverte peut être résumé, comme étant une recherche de l'ensemble des motifs ou des modèles appartenant à $\mathcal{L}$ et qui sont validés par le prédicat $\mathcal{P}$ en se basant sur les contraintes fixées par l'utilisateur. Cette discipline a été largement investie par différents domaines. À titre d'exemple, l'apprentissage supervisé, qui est un problème très connu de la fouille de données peut être représenté comme un problème de découverte de modèles à base de contraintes. La solution à chercher, est une fonction (un modèle de classification) h^* qui satisfait le prédicat de sélection $\mathcal{P}$ (c.-à-d. un prédicat qui sélectionne la fonction la plus performante en classant un sous-ensemble indépendant de la base de données $\mathcal{D}$) parmi toutes les solutions générées à partir d'autres sous-ensembles de la base de données $\mathcal{D}$ possibles qui forment le langage $\mathcal{L}$.

Enfin, beaucoup de techniques de fouille de données peuvent être représentées comme étant un problème de découverte de motifs ou modèles intéressants à base de contraintes, nous citons principalement la construction d'arbres de décisions, le clustering (Constrained Clustering), les SVM (Support Vector Machines).

2.2.1 L'espace de recherche

Pour la découverte des motifs intéressants, l'espace de recherche est le concept le plus important. Les éléments (c.-à-d. les sous-graphes connexes et les arbres de décisions (notés pour des cas généraux concepts, motifs)) de l'espace de recherche sont généralement menés d'une relation d'ordre. Cette relation d'ordre entre les motifs permet de définir un ordre efficace d'exploration de l'espace de recherche qui permet la restriction de cet espace en évitant l'exploration des motifs non intéressants (sous graphes connexes non fréquents, et des arbres de décisions non performants).

Relation d'ordre

Soit une relation d'ordre partiel $\preceq$ sur les motifs de l'espace de recherche $\mathcal{L}$. Un motif φ est plus général qu'un autre motif θ ssi $\varphi \preceq \theta$. De façon équivalente, nous statuons que θ est plus spécifique que φ. Par ailleurs, si aucune relation n'existe entre φ et θ, les deux motifs sont signalés incomparable. De plus, cette relation d'ordre (c.-à-d. spécialisation)$\preceq$ est une relation monotone en accord avec le prédicat $\mathcal{P}$ si pour une base de données $\mathcal{D}$: si $\varphi \preceq \theta$ et $P(\mathcal{D}, \theta) =$ vrai alors $\mathcal{P}(\mathcal{D}, \varphi) =$vrai.

Example 2.2.1 *Soit l'ensemble* $X = \{a, b, c, d, e\}$ *et* $P = (X, \preceq)$ *un ensemble ordonné, avec* $\preceq$ *une relation définie sur* X *comme suit :* $\preceq = \{(a, b),\ (a, e),\ (c, b),\ (c, d),\ (c, e),\ (d, e),\ (a, a),\ (b, b),\ (c, c),\ (d, d),\ (e, e)\}$. *La figure 2.1 illustre la représentation graphique des relations au sein de* $P = (X, \preceq)$.

Nous pouvons aussi représenter des relations au sein de $P = (X, \preceq)$ au moyen d'un graphique appelé diagramme de Hasse en utilisant que les couples de couvertures (c.-à-d. le successeur et le prédécesseur immédiat). À titre d'exemple la figure 2.2 illustre le diagramme de Hasse correspondant à l'ensemble ordonné défini sur l'exemple 2.2.1.

Motifs minimaux et maximaux

Soit une relation d'ordre $\preceq$ sur $\mathcal{L}$, et soit $\mathcal{B} \subseteq \mathcal{L}$ un ensemble de motifs, $min(\mathcal{B})$ est défini comme l'ensemble des motifs les plus généraux de $\mathcal{B}$.

$$min(\mathcal{B}) = \{\varphi \in \mathcal{B} | \nexists \theta \in \mathcal{B}\ t.q.\ \theta \prec \varphi\}$$

Donc pour un $\mathcal{B}$ le sous-ensemble de motifs qui ne sont pas fréquents de $\mathcal{L}$, $min(\mathcal{B})$ est le sous-ensemble des motifs non fréquents les plus généraux et nous notons par minimal tout motif de cet ensemble. De même, pour l'ensemble des motifs fréquents $\mathcal{F} \subseteq \mathcal{L}$, l'ensemble max($\mathcal{F}$) des motifs les plus spécifiques de $\mathcal{F}$.

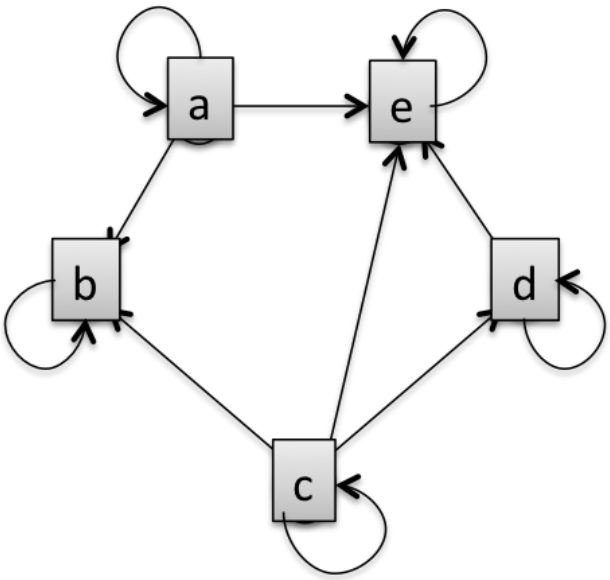

FIGURE 2.1 – Exemple graphique d'un ordre partiel.

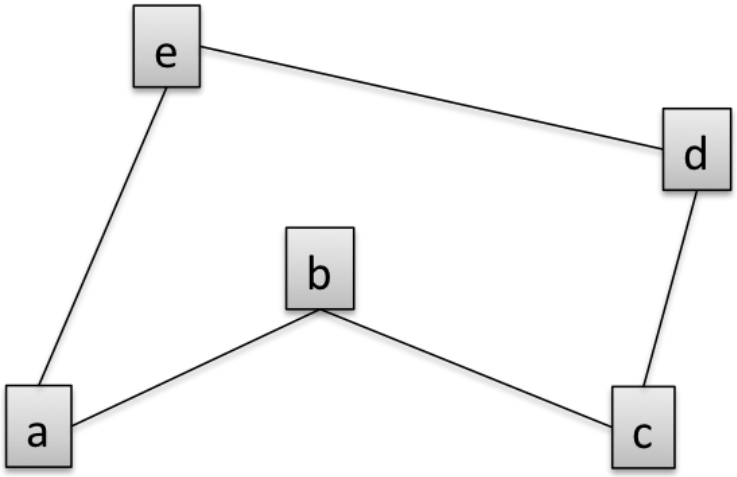

FIGURE 2.2 – Exemple d'un diagramme de Hasse.

$$\max(\mathcal{F}) = \{\varphi \in \mathcal{F} | \nexists \theta \in \mathcal{F} \; t.q. \; \varphi \prec \theta\}$$

Motifs successeurs et prédécesseurs immédiats

Étant donné un langage de motifs $\mathcal{L}$, un opérateur de spécialisation P_s et un autre de généralisation P_g associent à chaque motif φ les ensembles $P_s(\varphi)$ et $P_g(\varphi)$ composés respectivement des motifs spécialisant et généralisant φ, $P_s(\varphi) = \{\theta \in \mathcal{L} \mid \varphi \prec \theta\}$ et $P_g(\varphi) = \{\theta \in \mathcal{L} | \theta \prec \varphi\}$. Un motif d de P_s ou P_g est dit direct ou immédiat s'il est l'un

des motifs les plus généraux de P_s c.-à-d. $d \in min(P_s(\varphi))$ ou l'un des motifs les plus spécifiques de P_g c.-à-d. $v \in max(P_g(\varphi))$.

Catégorisation d'un espace de recherche

Comme nous l'avons mentionné auparavant l'efficacité de la recherche des solutions (motifs intéressants) dépend des propriétés de l'espace de recherche. Dans cette sous-section nous définissons quelques propriétés que nous allons rencontrer par la suite pour un espace de recherche $\mathcal{L}$ qui peut être appelé :

- Accessible si pour tout $X \in \mathcal{L} \backslash \{\emptyset\}$ il y a un $e \in X$ tel que $X \backslash \{e\} \in \mathcal{L}$;
- Fortement accessible si pour tout $X, Y \in \mathcal{L}$ satisfaisant $X \subset Y$, il y a un $e \in Y \backslash X$ tel que $X \cup \{e\} \in \mathcal{L}$;
- Indépendance si $Y \in \mathcal{L}$ et $\forall X \subseteq Y \rightarrow X \in \mathcal{L}$;
- Confluent (losange) si $\forall I, X, Y \in \mathcal{L}$ avec $\emptyset \neq I \subseteq X$ et $I \subseteq Y$, alors il existe $X \cup Y \in \mathcal{L}$.
- Treillis si pour tout couple $X, Y \in \mathcal{L}$ il existe un élément minimal $Z \in \mathcal{L}$ (Z=$X \wedge Y$) et un autre maximal $W \in \mathcal{L}$ (W=$X \vee Y$).

2.2.2 La découverte des motifs fréquents

Pour la découverte de motifs fréquents, en respectant un seuil δ défini par un utilisateur, un prédicat de sélection $\mathcal{P} : \mathcal{L} \rightarrow \{vrai, faux\}$ marque un motif $x \in \mathcal{L}$ fréquent si et seulement si $\mathcal{P} = vrai$ pour $|\mathcal{D}[x]| \geq \delta$ ($\mathcal{D}[x]$ est le sous-ensemble de motifs contenant x).

Definition 2.2.2 *(La découverte des motifs fréquents). Soit $\mathcal{D}$ une base de données, le support d'un motif X dans $\mathcal{D}$, noté $sup(X, \mathcal{D})$ est le nombre d'objets de $\mathcal{D}$ contenant X. Alors que, la fréquence de X, notée $freq(X, \mathcal{D})$, correspond au ratio des objets de $\mathcal{D}$ contenant X, soit*

$$freq(X, \mathcal{D}) = \frac{sup(X, \mathcal{D})}{|\mathcal{D}|}$$

Étant donné un seuil minimal de fréquence δ, le problème consiste à trouver un ensemble $\mathcal{F}$ de motifs fréquents (leurs fréquences dans $\mathcal{D}$ est supérieur à δ).

$$\mathcal{F} = \{X \subseteq I | freq(X, \mathcal{D}) \geq \delta\}.$$

Definition 2.2.3 *(Motif fréquent fermé) Soit $\mathcal{D}$ une base des données, X un motif fréquent, le motif $X \in \mathcal{F}$ est fermé ssi :*

$$\forall\ Y \in \mathcal{F}\ et\ X \subset Y\ \ freq(X, \mathcal{D})\ >\ freq(Y, \mathcal{D}).$$

Definition 2.2.4 *(Motif fréquent maximal) Soit $\mathcal{D}$ une base des données, X un motif fréquent, le motif $X \in \mathcal{F}$ est maximal ssi :*

$$\forall\, Y \in \mathcal{L},\ Y \notin \mathcal{F} \text{ et } X \subset Y.$$

Definition 2.2.5 *(La fermeture descendante) Soit un espace de recherche défini par une relation d'ordre $\preceq$ et soit θ un motif de cet espace, θ est fermé vers le bas, par rapport à $\mathcal{P}$, ssi tout motif $\varphi \prec \theta$ est intéressant selon $\mathcal{P}$.*

L'espace de recherche $\mathcal{L}$ induit par cette relation d'ordre, prenant souvent la forme d'un treillis, peut se représenter comme un graphe dirigé acyclique dont les noeuds sont les motifs de $\mathcal{L}$, et les arcs correspondent à des raffinements. La figure 2.3 illustre un exemple de l'espace de recherche des sous-ensembles d'items de l'ensemble $X = \{0, 1, 2, 3\}$.

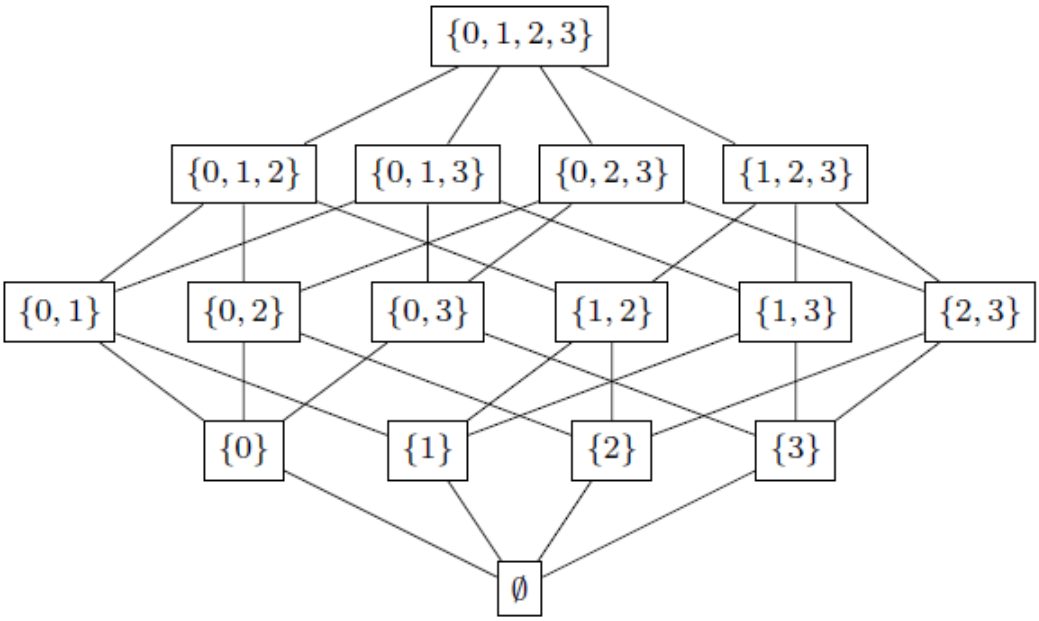

FIGURE 2.3 – l'espace de recherche des sous-ensembles d'items de l'ensemble $X = \{0, 1, 2, 3\}$.

Généralement, le problème de découverte des motifs fréquents se résume en deux tâches principales : 1) La génération des candidats ; 2) L'énumération des candidats fréquents. Ce processus est d'une apparence facile, mais beaucoup de difficultés sont liées à ces deux étapes. Pour la première tâche, nous citons principalement (i) la manière de génération des candidats et (ii) la redondance des candidats. La deuxième étape est corrélativement liée (i) à la complexité des motifs et (ii) le nombre de fois que la base de données est

parcourue afin de calculer la fréquence de tous les candidats. Par conséquent, la difficulté du problème est fortement liée à la structure de donnée manipulée (c.-à-d. le type de motif) item, séquence, arbre, graphe, etc.

La complexité de l'espace de recherche c.-à-d. (le langage $\mathcal{L}$) est un autre problème très influant par rapport à la stratégie d'exploration de l'espace de recherche. La richesse de cet espace de recherche $\mathcal{L}$ peut induire un espace de motifs (solutions) infini, en accord avec un prédicat $\mathcal{P}$ (fonction objective) satisfait par une infinité d'éléments de $\mathcal{L}$, l'ensemble des motifs intéressants $Th(\mathcal{D}, \mathcal{L}, \mathcal{P})$ (solutions) ne peut jamais être déterminé. Cependant, même avec un espace $\mathcal{L}$ restreint, l'efficacité de la recherche des motifs intéressants (solutions) dépend fortement de la présence d'un ensemble de propriétés définissant les relations entre les motifs de $\mathcal{L}$. Le fait d'avoir un ordre partiel entre ces motifs fournit une base solide pour l'organisation de la recherche. À titre d'exemple pour les itemsets, la relation " incluse ou égale $\subseteq$" décrit des liens entre certains ensembles d'items, ce qui a permis de nombreuses possibilités de manoeuvres l'amélioration de l'efficacité de la recherche.

2.2.3 Opérateur de fermeture

Un autre concept que nous allons évoquer est l'opérateur de clôture (fermeture). Un opérateur de clôture sur un ensemble ordonné $(\mathcal{L}, \preceq)$ est une application $\sigma : \mathcal{L} \rightarrow \mathcal{L}$ vérifiant les trois propriétés suivantes (pour $\forall x, y$ de $\mathcal{L}$) :

- $x \preceq \sigma(x)$ (σ est extensive) ;
- si $x \preceq y$ alors $\sigma(x) \subseteq \sigma(y)$ (σ est croissante) ;
- $\sigma(\sigma(x)) = \sigma(x)$ (σ est idempotente).

En projetant cette dernière définition pour le cas de la découverte des motifs fréquents, les motifs fermés représentent les points fixes qui satisfont que σ soit idempotente. Il est clair que tous les motifs qui admettent un fils (un successeur ou une spécialisation) qui est équivalent en terme de fréquence ne sont pas fermés.

Sachant la fermeture descendante de l'espace de recherche $\mathcal{L}$, les motifs fermés sont les motifs pour lesquels tous les motifs immédiats affichent une baisse de fréquence : $\forall x \in \mathcal{C}, y \in \mathcal{L}$, et $x \subset y$ $\mathcal{D}[x] \supset \mathcal{D}[y]$. Nous noterons un graphe i comme un graphe inductif générateur si et seulement s'il existe deux graphes fermés $c_1 \subset c_2 \in \mathcal{C}$ tel que $i = c_1 \cup \{e\}$ et $c_2 = \sigma(i)$.

2.3 La théorie des graphes : quelques notions de bases

Depuis plusieurs années, la théorie des graphes est l'un des domaines les plus populaires des mathématiques et de l'informatique. Cette popularité est due aux nombreuses applications dans différents domaines telles que la biologie, la chimie, l'électronique, etc. Un graphe permet la représentation d'un ensemble complexe d'objets en exprimant les relations entre eux ; circuits électroniques, réseaux routiers, les molécules, etc.

La complexité structurelle des graphes est liée au fait qu'un graphe permet d'encoder deux informations : l'information représentée par les sommets, ainsi que celle représentée par les arêtes. Un graphe G est un couple d'ensembles $V(G) = \{v1, v2, v3, ...\}$, et $E(G) = \{e1, e2, e3, ...\}$. Les éléments du premier ensemble sont notés sommets (ils sont aussi appelés points ou noeuds) et ceux du deuxième ensemble sont notés les arêtes.

Definition 2.3.1 *(Un graphe)* $G = (V, E)$ *se compose d'un ensemble de sommets* V *et d'un ensemble d'arêtes* $E \subseteq V \times V$ *où chaque arête permet de relier deux sommets. De façon équivalente, on dit que* $V(G)$ *et* $E(G)$ *sont des ensembles contenant respectivement les sommets et les arêtes du graphe* G*.*

Un graphe est dirigé (orienté), si ses arêtes possèdent un ordre d'orientation entre les deux sommets de chacune, des arêtes c.-à-d. un sommet initial u et un sommet terminal v, c.-à-d. $(u, v) \neq (v, u)$. Alors qu'un graphe dont deux sommets sont reliés par plusieurs arêtes est un multigraphe. Un graphe ayant une arête reliant un sommet à lui-même est un pseudo-graphe. En plus, un graphe qui est ni un multigraphe, ni un pseudo graphe est signé comme un graphe simple. La figure 2.4 illustre un exemple d'un : (a) graphe simple non orienté, (b) graphe simple orienté, (c) pseudo graphe, et (d) multigraphe.

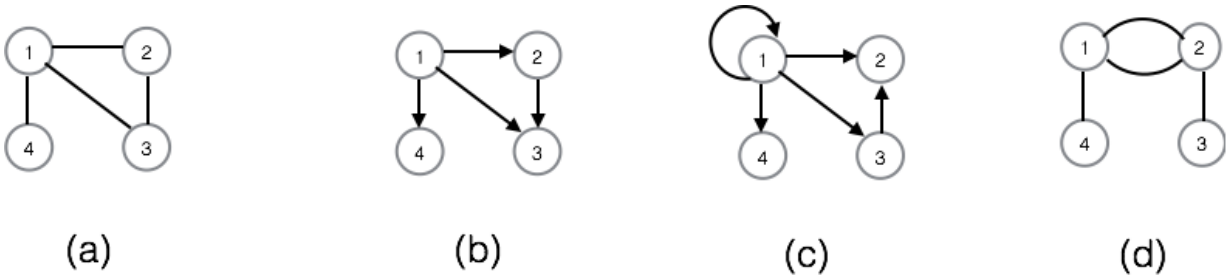

FIGURE 2.4 – Quelques exemples de graphes (1)

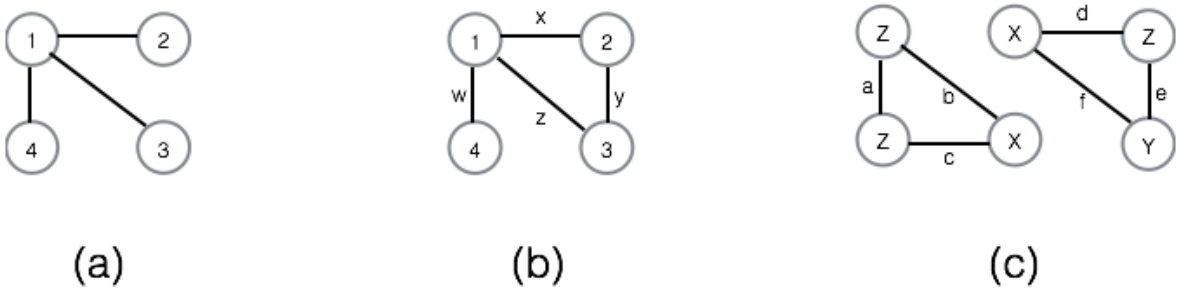

FIGURE 2.5 – Quelques exemples de graphes (2)

Un graphe est noté étiqueté $G(V, E, \Sigma, L)$ en lui associant un ensemble Σ d'étiquettes et une fonction $L : V \cup E \longrightarrow \Sigma$ qui assigne les étiquettes $e_t \in \Sigma$ aux ensembles d'arêtes et de sommets V et E. Une chaîne est une séquence de q arêtes $e_1,\ e_2, ...,\ e_q$, pour $i > 1$ chaque couple d'arêtes e_i et e_{i-1} partagent un sommet en commun. S'il existe un sommet en commun entre la dernière et la première arête e_1 et e_q, la chaine est fermée et sera notée cycle.

Un graphe sans cycle est un arbre. Un graphe G est connexe s'il existe une chaine entre toute paire de sommets. Un graphe G est un graphe à étiquetage d'arêtes non redondant si et seulement si l'étiquette de chaque arête survient au plus une fois. La figure 2.5 illustre un exemple d'un : (a) arbre simple, (b) graphe simple étiqueté, (c) graphe non connexe, et (b, c) des graphes ayant des arêtes non redondantes.

Conclusion du chapitre

Dans ce chapitre, nous avons présenté les notions préliminaires de la théorie des graphes. Nous avons illustré les concepts et les fondements de base de la découverte des motifs intéressants à base de contraintes. Ainsi que l'espace de recherche et les particularités liées à ce dernier. En particulier, nous avons illustré la relation d'ordre entre les motifs, et nous avons aussi illustré que beaucoup de problèmes de la fouille de données comme la construction d'un arbre de décision, les SVM, et le clustering ne sont en pratique que des spécialisations de la découverte des modèles intéressants, où l'objectif est de trouver le modèle (le sous-arbre de décision) le plus performant parmi tous les modèles (sous arbres de décisions) possibles.

Chapitre 3

*R*EVUE DE LA LITTÉRATURE

Sommaire

Ce chapitre introduit deux spécialisations de la fouille des motifs à base de contraintes (c.-à-d. constraint-based pattern mining) [Nijssen2010a]. D'une part, la fouille des motifs et des sous-graphes connexes fréquents. Les principaux travaux et résultats de la littérature portant sur la découverte des sous-graphes connexes fréquents vont être présentés. Et d'autre part, la recherche d'un arbre de décision optimal dans l'espace de recherche d'arbres.

3.1 La découvertes des motifs fréquents

Depuis le début des années 90, de nombreux chercheurs se sont intéressés à l'extraction de motifs fréquents. Les motifs fréquents sont typiquement des sous-structures (c.-à-d. des sous-ensembles d'attributs (itemsets), des sous-arbres, des sous-graphes, etc.), dont le nombre d'occurrences est significatif (c.-à-d. supérieur à un seuil donné) par rapport à une base des données transactionnelles donnée.

3.1.1 Itemsets

La découverte des itemsets fréquents est une spécialisation très populaire de la découverte des motifs fréquents. Depuis une vingtaine d'années, cette discipline a été la base d'un nombre important d'approches de la fouille de données comme les règles d'associations [Mannila and Toivonen1997], [Gunopulos *et al.*1997], la sélection d'attributs [Kotsiantis2013], [Kohavi and John1997], etc. Le problème de découverte des itemsets fréquents peut être défini comme suit : soit une base de données transactionnelle d'items, l'objectif est de trouver tous les itemsets fréquents ayant une occurrence au niveau d'au moins δ transactions avec δ un seuil minimum défini par l'utilisateur.

Comme tout processus de découverte de motifs, la recherche des itemsets fréquents consiste à explorer l'espace de recherche de tous les sous-ensembles d'items possibles pour extraire que les fréquents entre eux. Cette tâche nécessite principalement la génération des sous-ensembles (c.-à-d. les candidats) pour lesquels la fréquence est testée par la suite. Autrement dit, la tâche consiste à commencer à partir du vide $\emptyset$ qui est un sous-ensemble de toutes les transactions pour une base de données non redondante sinon le commencement est à partir de l'ensemble d'items qui est inclus sur toutes les transactions de la base de données. Ensuite, en utilisant un opérateur de spécialisation, les sous-ensembles d'items les plus spécifiques sont explorés. Un candidat (c.-à-d. un sous-ensemble d'items) plus spécifique est généré en ajoutant un nouvel attribut à un sous-ensemble d'items qui a déjà été vérifié fréquent.

Les principales difficultés rencontrées pour la découverte des itemsets sont la redondance des candidats lors de la génération, et le nombre d'accès à la base de données pour calculer la fréquence. Contrairement aux cas de motifs plus complexes, la notion d'identité unique des items au niveau des itemsets permet de résoudre ce problème par la limitation : 1) des possibilités de jointures en respectant un unique ordre de comparabilité (ordre lexicographique) entre les ensembles d'items candidats de jointure pour les méthodes à l'apriori [Agrawal *et al.*1994] ; 2) des possibilités d'augmentations, en respectant l'ordre

lexicographique entre les items pour les méthodes en profondeur [Grahne and Zhu2005]. Cette limitation de possibilités revient à réduire le nombre d'accès à la base de données.

Example 3.1.1 *L'un des premiers algorithmes de résolution du problème d'itemsets est "APRIORI", ce dernier a été proposé par [Agrawal* et al.*1994]. Cet algorithme considère une base de données $\mathcal{D}$ en entrée, ainsi qu'un seuil minimum de fréquence δ, et vise la récupération de tous les sous-ensembles fréquents d'items dans $\mathcal{D}$ qui ont apparu au moins au sein de δ ensembles de $\mathcal{D}$, à l'aide d'une recherche par niveau (c.-à-d. en largeur). En commençant par le premier niveau avec un ensemble qui inclut tous les sous-ensembles d'items candidats n_1, chacun des sous-ensembles est composé d'un unique item. Le support (fréquence) des ensembles candidats est obtenu en explorant la base de données et en calculant le nombre de transactions (ensembles d'items) pour lesquelles ils sont inclus. Les ensembles candidats qui sont fréquents, et dont le support est au moins δ sont alors ajoutés à l'ensemble des sous-ensembles d'items fréquents $\mathcal{F}$. Les ensembles candidats du niveau $k+1$, (c.-à-d. n_{k+1}) sont ensuite générés en combinant (c.-à-d. jonction) deux ensembles fréquents X et Y de k items, qui ne se distinguent que d'un seul item. Afin d'éviter de générer plus qu'une fois un même candidat, seules les jonctions pour lesquelles $X_k < Y_k$ sont considérées. Ensuite, tous les candidats incluent un sous-ensemble non fréquent de k items, sont élagués. Cette phase d'élagage est appliquée afin de respecter la propriété de fermeture descendante. Enfin, pour chacun des candidats de n_{k+1} la base de données est parcourue afin de compter le nombre de transactions l'incluant. Seuls les candidats, dont le support est au moins δ. Ce processus est répété jusqu'à ce qu'il n'y ait plus de nouveaux candidats (c.-à-d. lorsque $n_k = \emptyset$).*

3.1.2 La découverte des sous-arbres fréquents

Ces dernières années avec l'explosion de l'utilisation du data mining. Les données transactionnelles ne satisfont plus le besoin de représenter de nouveaux types de données. Beaucoup de documents semi-structurés (par exemple HTML, LaTeX, SGML, BibTeX, etc.) sont de plus en plus utilisés pour représenter la connaissance. De manière générale, les documents semi-structurés sont naturellement hiérarchiques, et les informations peuvent être efficacement représentées comme des structures arborescentes.

Généralement, le processus de découverte des sous-arbres fréquents est plus coûteux que celui de la découverte des itemsets. Ce phénomène est dû à la complexité de génération des candidats. La génération d'un nouveau sous-arbre à partir d'un sous-arbre déjà validé fréquent, nécessite l'ajout d'un nouveau sommet en le reliant à un autre déjà existant par une arête. Soit $V(T)$ et $ETE(T)$ l'ensemble des sommets et des étiquettes

d'arêtes d'un arbre T respectivement, et $V(t)$ et $ETE(t)$ l'ensemble des sommets et des étiquettes d'arêtes d'un sous arbre t de T il y a $\mid |V(T)/V(t)| * |ETE| \mid$ augmentations possibles à partir de chacun des sommets $|V(t)|$ du sous-arbre t. Pour le cas des arbres étiquetés, la tâche d'augmentation est encore plus complexe, car il faut aussi prendre en compte les possibilités d'étiquetages des arêtes et des sommets. De plus, afin d'éviter la génération des mêmes candidats chacun des nouveaux candidats générés est comparé avec d'autres qui ont été déjà visités à l'aide d'un test d'isomorphisme d'arbres qui est de loin plus complexe que le cas des itemsets.

Pour tester la fréquence des sous-arbres candidats, autre que le nombre d'accès à la base de données, le test d'isomorphisme de sous arbre utilisé est généralement d'une complexité polynomiale, mais largement plus complexe que de tester la relation de sous ensemble d'items. En contrepartie, l'énumération reste globalement à un délai polynomial, mais les problèmes de redondance et de fréquence sont de difficultés variantes selon le type d'arbres manipulés. Afin de contourner le problème de redondance (c.-à-d. d'isomorphisme d'arbres), beaucoup de travaux ont proposé un encodage unique pour chaque arbre appelé code canonique. La complexité d'encodage canonique diffère entre les arbres ordonnés et ceux non ordonnés [Chi *et al.*2005].

À titre d'exemple, il est facile de remarquer qu'un arbre simple nécessite plus d'attention pour l'encoder qu'un arbre ordonné, mais le nombre d'arbres simples est plus grand de ceux ordonnés. Par conséquent, les possibilités d'augmentations (la génération d'un arbre de taille plus un) sont plus grandes pour le cas d'un arbre simple que celui d'un arbre ordonné, donc le coût de génération des candidats est plus élevé pour le cas d'un arbre simple. Pour calculer la fréquence, le test d'isomorphisme de sous-arbres simples coût au pire des cas $\mathcal{O}(|V(t)| \ |E(t)|)$ alors que $\mathcal{O}(\frac{|V(t)| \ |E(t)|^{1/2}}{log \ (|E(t)|)})$ suffit pour le cas d'un arbre ordonné [Nijssen2010b].

3.2 La découverte des sous-graphes connexes fréquents

Comme nous l'avons déjà mentionné, la découverte des sous-graphes connexes fréquents est une autre instance très importante de la découverte des motifs intéressants. Motivé par les nombreuses applications auxquelles les données sont représentées comme des graphes. Contrairement aux problèmes de découverte des itemsets et des sous-arbres fréquents où juste un délai polynomial est nécessaire pour l'énumération de deux fréquents successifs, le problème pour les graphes est beaucoup plus difficile et peut même être intraitable.

Le problème de découverte des sous-graphes connexes fréquents peut être défini de deux différentes façons :

1. Soit un ensemble de graphes $\mathcal{D}_G$, le but est de récupérer tous les sous-graphes connexes qui sont présents au moins sur δ graphes $G_i \in \mathcal{D}_G$.
2. Soit un seul grand graphe $\mathcal{G}$, le but est de récupérer tous les sous-graphes qui occurrent au moins δ fois dans ce grand graphe.

De façon générale, pour les deux cas un algorithme de découverte des sous-graphes connexes fréquents doit faire face principalement à l'isomorphisme de sous graphes pour tester la fréquence, ainsi que l'isomorphisme de graphes afin d'éviter la génération de duplicatas. Encore, cet algorithme doit aussi faire attention à l'optimisation de nombre d'augmentations (la jonction pour les algorithmes à l'apriori [Agrawal *et al.*1994]) (c.-à-d. ajouter une arête ou un sommet à un sous-graphe fréquent pour créer un nouveau sous-graphe connexe candidat). Pour le cas d'un seul grand graphe, il faut être très attentif par rapport à la propriété de fermeture descendante du support, parce qu'en cas de présence de chevauchement entre des plongements différents (sous graphes différents qui partagent au moins une même arête ou un même sommet) cette propriété sera violée. Il existe beaucoup de méthodes de découverte des sous-graphes fréquents qui peuvent être catégorisées selon : 1) la stratégie d'exploration de l'espace de recherche, et 2) la façon dont ces méthodes gèrent les problèmes d'isomorphisme de graphes et de sous-graphes.

Pour le cas d'un seul grand graphe, comme nous l'avons évoqué précédemment, les algorithmes de découvertes sont concernés par une tâche supplémentaire pour préserver la fermeture descendante du support. Nous citons deux algorithmes très connus : HSIGRAM et VSIGRAM [Kuramochi and Karypis2005]. Ces deux algorithmes définissent le support comme la taille de l'ensemble d'indépendances maximales des sous-graphes chevauchés, ce qui permet de préserver la propriété de fermeture descendante. Ces deux algorithmes parcourent l'espace de recherche de deux manières différentes : soit en largeur pour le premier et en profondeur pour le deuxième. Il existe d'autres algorithmes pour la découverte des sous-graphes connexes au sien d'un seul grand graphe [Fiedler and Borgelt]. Par ailleurs, le cas de découverte des sous-graphes connexes fréquents par rapport à une base de graphes est plus populaire et beaucoup de techniques ont été proposées, nous citons principalement : [Thomas *et al.*2010], [Huan *et al.*2004], [Huan *et al.*2003], [Inokuchi *et al.*2000], [Kuramochi and Karypis2001], [Yan and Han2002], [Nijssen and Kok2005]. Ces algorithmes explorent l'espace de recherche de différentes manières en largeur et en profondeur, ils utilisent des techniques exactes ainsi que d'autres approchées, soit pour la restriction de

l'espace de recherche, et/ou même pour résoudre le problème d'isomorphisme de sous-graphes, et suivent aussi différents schémas de canonisation pour tester l'isomorphisme de graphes afin d'éviter la génération de candidats redondants. Dans la suite de ce manuscrit, nous allons considérer que le cas de découverte des sous-graphes connexes fréquents sur une base de graphes.

3.3 Les problèmes de découverte des sous-graphes fréquents

Comme tout algorithme de découverte des motifs fréquents, les principales tâches de la découverte des sous-graphes connexes fréquents sont la génération des candidats, et le calcul de la fréquence. Malheureusement, ces tâches sont beaucoup plus difficiles en raison de la complexité structurelle des graphes. La résolution de ces deux tâches consiste en la résolution d'isomorphisme de graphes et de sous-graphes pour chacun des candidats explorés, sachant qu'ils sont deux problèmes difficiles. La tâche de génération d'un sous-graphe plus spécifique à partir d'un plus général nécessite un nombre important de tests d'augmentations en raison du grand nombre de possibilités que cela soit de relier par une nouvelle arête deux sommets déjà existants, ou d'ajouter un nouveau sommet et le relier avec l'un des sommets déjà existant.

3.3.1 La génération des sous-graphes connexes candidats

La génération d'un nouveau sous-graphe connexe à k-sommets (k-arêtes) consiste généralement à le produire : 1) à partir d'un autre à k-1-sommets (k-1-arêtes) en lui rajoutant un sommet (une arête), ou un sous-graphe à k+1-sommets (k+1-arêtes) en éliminant un sommet (une arête). 2) à partir d'une jonction de deux sous-graphes à k-1-sommets (k-1-arêtes) qui partagent un sous-graphe connexe en commun à k-2-sommets (k-2-arêtes) par l'union de leurs sommets et arêtes. Et 3) Même à partir de l'intersection d'un ensemble de graphes $G_i \in \mathcal{D}$ qui contiennent un certain sous-graphe générateur inductif g. Cette opération d'intersection produit le sous-graphe maximal en commun qui peut être non connexe, alors l'ensemble de ces sous-graphes connexes maximaux est récupéré. Cette dernière technique de génération est beaucoup plus coûteuse, mais elle permet de réduire considérablement le nombre des candidats à explorer.

L'augmentation :

Cette opération d'augmentation consiste à ajouter une arête ou un sommet à un sous-graphe à k-1-arêtes(k-1-sommets) connexe qui a déjà été exploré afin de construire un

nouveau sous-graphe connexe à k-arêtes (k-sommets). Contrairement au cas des itemsets où la tâche d'augmentation se résume à ajouter un nouvel item à un ensemble d'items fréquent déjà exploré, pour les graphes connexes, cette opération d'ajouter soit un sommet ou une arête est beaucoup plus compliquée. La difficulté de cette opération à l'égard des graphes est due au nombre important de possibilités de jonctions. Afin d'ajouter un sommet, tous les sommets du graphe sont des cibles potentielles pour les lier au nouveau sommet. De plus, il en est de même pour le choix du sommet à relier au nouveau sommet. Une étiquette d'arête parmi toutes les étiquettes possibles doit être associée à ce nouveau lien. Pour ajouter une nouvelle arête, l'opération d'augmentation consiste à chercher deux sommets qui partagent les mêmes étiquettes des sommets de l'arête à ajouter et les lier. S'il n'existe qu'un seul des deux sommets alors un nouveau sommet est ajouté avec l'arête au sous-graphe.

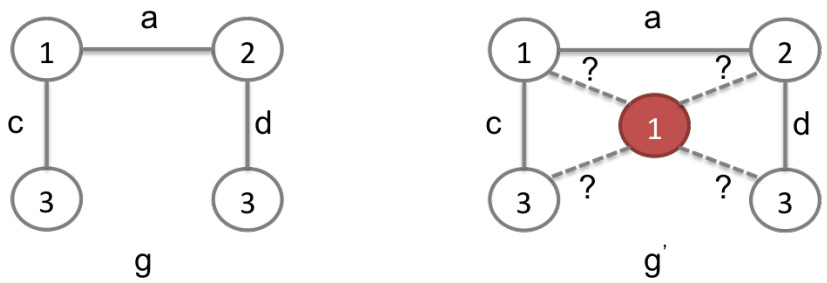

FIGURE 3.1 – Exemple d'augmentation d'un sous-graphe par l'ajout d'un sommet.

Pour l'ajout d'un nouveau sommet, il est possible de le relier à chacun des sommets du sous-graphe. De même pour le cas de l'ajout d'une arête où il peut y avoir un nombre important de sommets ayant le même étiquetage que celui de l'arête à ajouter. La figure 3.1 illustre un exemple d'augmentation d'un sous-graphe g par l'ajout d'un sommet, le graphe g' produit où ce nouveau sommet (rouge) avec l'étiquette 1 peut être relié à chacun des sommets du sous-graphe ce qui permet de générer jusqu'à $(2^4 - 1)$ sous-graphes connexes différents.

Alors que la figure 3.2 illustre un exemple par rapport à l'ajout d'une arête e au sous-graphe g. L'opération d'augmentation dans ce cas est réalisée soit en reliant deux sommets déjà existants soit en ajoutant un nouveau sommet. Par conséquent, comme l'illustre le graphe g' il est possible d'avoir 6 sous-graphes connexes différents à l'issue de cette augmentation.

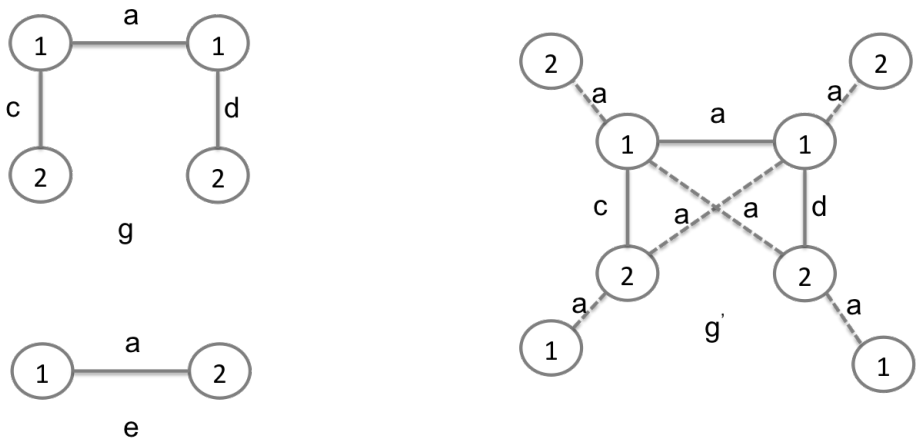

FIGURE 3.2 – Exemple d'augmentation d'un sous-graphe par l'ajout d'une arête.

La jonction :

Utilisée la première fois par [Agrawal *et al.*1994], la jonction est l'une des principales opérations caractéristiques de l'exploration à l'apriori. Cette opération de jonction consiste en l'union des composants de deux motifs d'un même niveau de spécification (c.-à-d. cardinal, taille, nombre de chemins disjoints, etc.) ayant un seul élément de différence (c.-à-d. item, sommet, arête, etc.) qui ont été déjà explorés et classés fréquents. Pour le cas des itemsets, la jonction est l'union des items de deux sous-ensembles de k-1-items à la différence d'un item, afin de générer un nouvel ensemble d'items à k-items. Alors que cette opération semble être simple pour le cas des itemsets, due à la notion d'identité, la jonction entre un couple de sous-graphes connexes fréquents à k-1-sommets (k-1-arêtes) est beaucoup plus complexe.

La jonction de deux sous-graphes à k-1-sommets (k-1-arêtes) nécessite qu'ils partagent un sous-graphe connexe de k-2-sommets (k-2-arêtes) ce qui est difficile à vérifier ($NP-complet$) sans avoir recours à de nombreux tests d'isomorphisme de sous-graphes. À partir du sous-graphe en commun, les deux sommets (arêtes) différents sont ajoutés, cependant est-ce que les deux sommets ajoutés seront reliés ou non ? S'ils sont considérés reliés, qu'elle est l'étiquette suggérée à cette arête ? Pour deux arêtes ajoutées ayant deux sommets avec la même étiquette, est-ce que les deux sommets seront fusionnés pour un seul sommet ou les considérer comme deux sommets différents ?

Pour le premier cas, soit deux sous-graphes connexes g_1 et g_2, pour $ETE(g_1)$ et $ETE(g_2)$ des ensembles contenant les étiquettes des arêtes de g_1 et g_2, $|ETE(g_1) \cup$

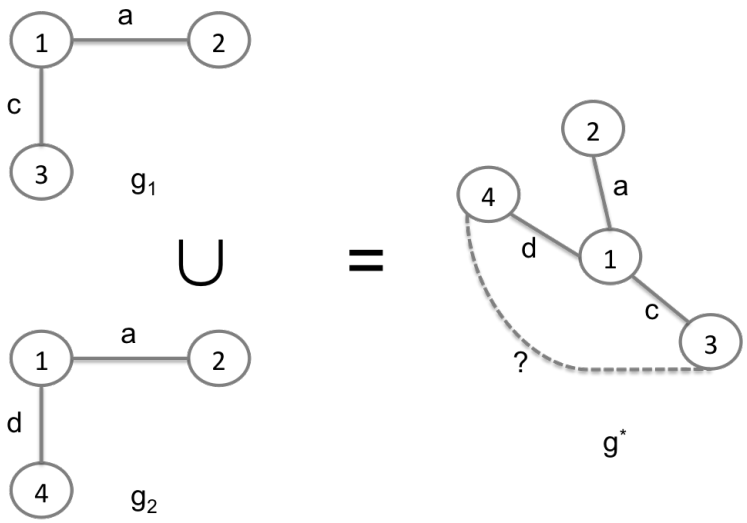

FIGURE 3.3 – Exemple de jonction de deux graphes connexes à base de sommets.

$ETE(g_2)| + 1$ est le nombre de valeurs possibles pour l'étiquette de l'arête entre les deux sommets différents (la valeur supplémentaire est pour le cas où les deux sommets ne sont pas reliés). La figure 3.3 illustre un exemple de jonction de deux sous-graphes g_1 et g_2 et g^* le nouveau sous-graphe généré qui peut avoir quatre valeurs peuvent être attribuées pour l'arête entre le sommet avec l'étiquette 3 et celui avec l'étiquette 4 (c.-à-d. il n'existe pas de lien, et les étiquettes a,c,d).

Pour le deuxième cas, soit deux sous-graphes connexes g_1 et g_2, la jonction à base d'arêtes produit deux sous-graphes connexes différents. La figure 3.4 illustre un exemple de jonction de deux sous-graphes g_1 , g_2 et $g^{'}$, $g^{''}$ les sous-graphes connexes produits.

La fermeture par intersection

Comme nous l'avons déjà évoqué, un processus de découverte des motifs fréquents souffre principalement d'explosion combinatoire. Alors que ce phénomène est dû à la fermeture descendante des motifs par rapport à la fréquence. En contrepartie, cette fermeture descendante permet de régénérer tous les motifs fréquents à partir de deux ensembles de motifs fréquents particuliers qui sont les motifs fermés et maximaux fréquents. Ces motifs fermés et maximaux fréquents sont exponentiellement peu nombreux que l'ensemble des motifs fréquents. Alors le fait de récupérer que ces motifs fermés et maximaux fréquents est beaucoup plus avantageux et permet d'achever la recherche pour un coût largement plus raisonnable, et que la taille des résultats permet une interprétation plus facile par la suite.

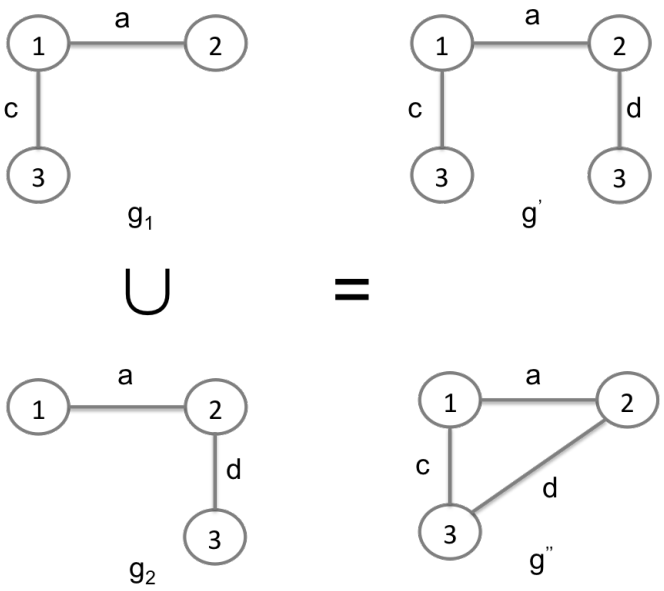

FIGURE 3.4 – Exemple de jonction de deux graphes connexes à base d'arêtes.

Pour profiter de cette orientation, il faut prendre en compte la réduction des candidats explorés. Pour le cas des itemsets fréquents fermés, l'indépendance de l'espace de recherche a été investie afin d'explorer qu'un ensemble de candidats d'ordre de $\mathcal{O}(\mathcal{C})$. La technique proposée afin d'atteindre ce résultat consiste à calculer la fermeture d'un ensemble d'items générateur inductif (c.-à-d. une augmentation d'un ensemble d'items fréquent fermé) qui permet de générer un nouveau sous-ensemble d'items fermé. L'opération de calcule de la fermeture d'un sous-ensemble d'items X consiste en l'intersection des ensembles d'items $\mathcal{I}_i \in \mathcal{D}$ tel que $X \subseteq \mathcal{I}_i$.

TABLE 3.1 – Exemple d'une base de transactions d'items.

L'étiquette	la transaction
T1	$\{0, 1, 3\}$
T2	$\{1, 2, 3\}$
T3	$\{1\}$
T4	$\{3\}$
T5	$\{1, 3\}$

Comme l'illustre la figure 3.5 par rapport à la base de transactions sur la Table 3.1 et un seuil de fréquence de 1, les sous-ensembles d'items fermés sont ceux encadrés par le

rouge (c.-à-d. $\{\emptyset\}$, $\{1\}$, $\{3\}$, $\{0,1,3\}$, $\{1,2,3\}$). Le sous-ensemble $\{\emptyset\}$ est toujours fermé tant que la base de données est non redondante (son taux d'apparition de 5) par augmentation vers les sous-ensembles $\{1\}$, et $\{3\}$ (taux d'apparition de 4), en calculant leurs fermetures (c.-à-d. $\cap\mathcal{D}[\{1\}]$, $\cap\mathcal{D}[\{3\}]$) les sous-ensembles d'items fermés fréquents $\{1\}$, et $\{3\}$ sont récupérés avec un taux d'apparence de 4. Alors qu'en augmentant le sous-ensemble $\{\emptyset\}$ vers $\{0\}$, et $\{2\}$ (taux d'apparition de 1), en calculant leurs fermetures, les sous-ensembles d'items fermés fréquents $\{0,1,3\}$, et $\{1,2,3\}$ sont récupérés avec un taux d'apparition de 1.

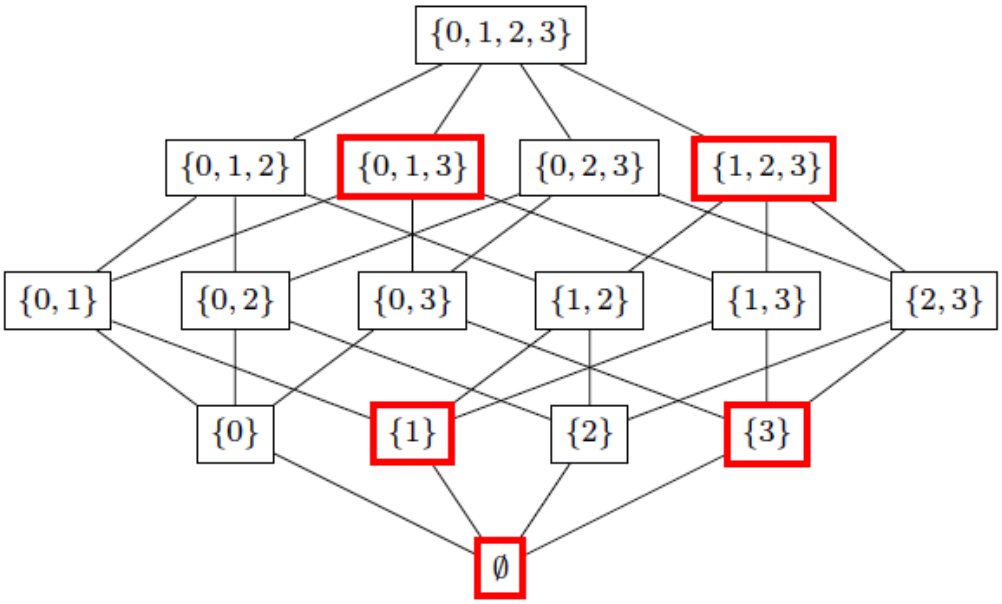

FIGURE 3.5 – l'espace de recherche des sous-ensembles d'items de l'ensemble d'items $X = \{0,1,2,3\}$.

Contrairement aux itemsets, l'application de l'intersection par rapport aux graphes connexes est beaucoup plus complexe. L'intersection peut être appliquée par l'intersection des ensembles de sommets et d'arêtes des graphes connexes. Cette intersection n'assure dans aucun cas la production d'un graphe connexe. Alors nous aurons recours à une exploration du graphe produit afin de tester s'il est bien connexe ? Si ce n'est pas le cas, un processus de découverte de composantes connexes maximales est exécuté pour récupérer tous les sous-graphes connexes maximaux du graphe non connexe produit pour lesquels les taux d'apparences n'affichent aucune baisse de fréquence par rapport à celui du sous-graphe générateur inductif. La figure 3.6 illustre un exemple de l'intersection de deux graphes connexes g_1 et g_2 qui produit un graphe non connexe g^*. L'exploration de

g^* permet la récupération de deux sous graphes connexes g' et g''.

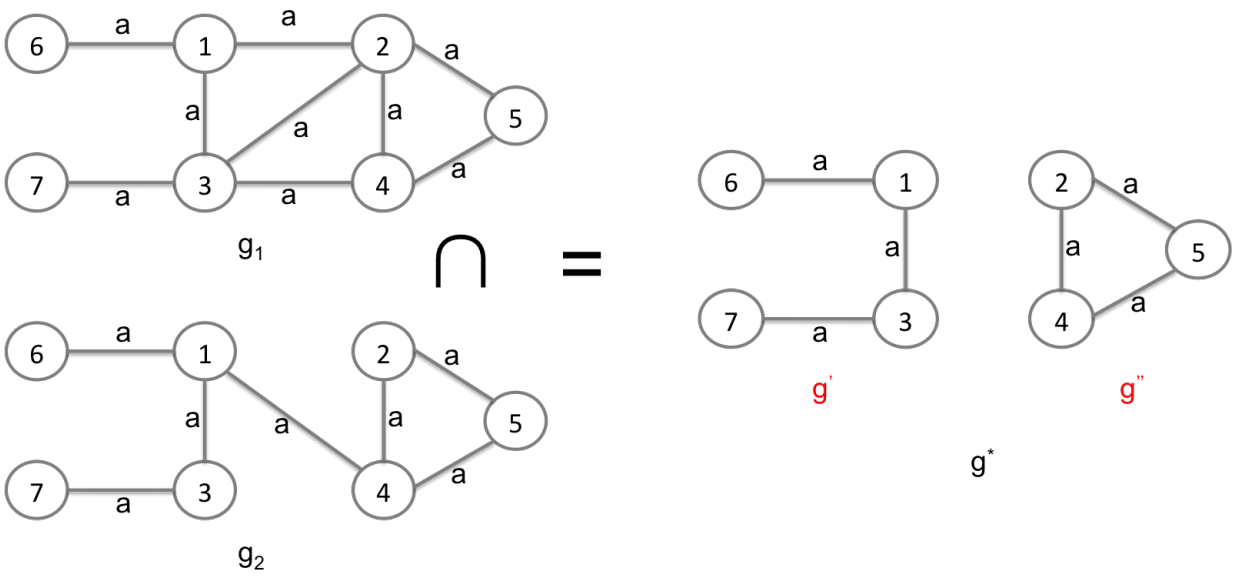

FIGURE 3.6 – Exemple d'intersection de deux graphes connexes.

Comme nous l'avons déjà mentionné, un autre problème important lié à la tâche de génération de motifs consiste à éviter la génération des redondants. Alors que le cas d'itemsets a nécessité de prendre en compte un ordre lexicographique, et un encodage d'ordre polynomial pour les arbres. Pour le cas des graphes, nous aurons affaire à un test d'isomorphisme connu difficile.

Le problème d'isomorphisme de graphes

Deux graphes $G_1(V_1, E_1)$, $G_2(V_2, E_2)$ sont isomorphe l'un vers l'autre si et seulement s'il existe une bijection φ de V_1 vers V_2 tel que $(v_1, v_2) \in E_1$ si et seulement si $(f(v_1), f(v_2)) \in E_2$. Le problème de décision de l'isomorphisme de graphes dans le cas général est de complexité NP, mais à ce jour il n'est pas connu P ou NP-complet.

Example 3.3.1 *La figure 3.7 illustre un exemple de deux graphes connexes isomorphe. Les couples de sommets de même couleur représentent des sommets jumeaux.*

La difficulté de vérifier un isomorphisme entre deux graphes est due au fait que la plupart des techniques parcourant l'ensemble de l'arbre des possibilités (espace de recherche) en élaguant les branches que l'on sait bloquantes, ce qui est généralement très coûteux et

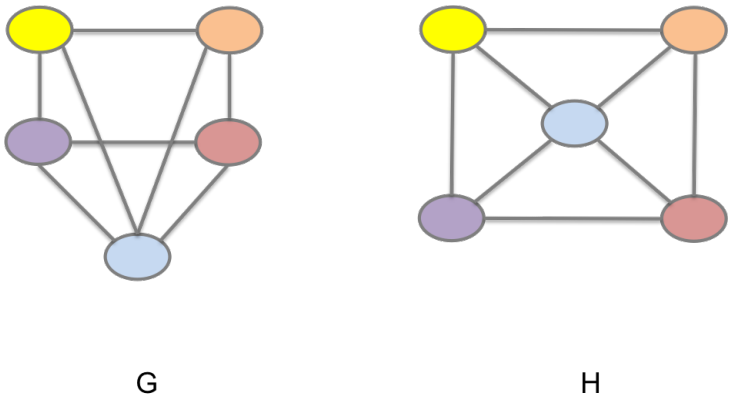

FIGURE 3.7 – Exemple d'isomorphisme de deux graphes connexes.

augmente corrélativement par rapport à la taille, le type et l'étiquetage des graphes d'entrées. Cependant, dans beaucoup de cas, trouver la représentation canonique d'une paire de graphes est équivalant à résoudre le problème d'isomorphisme de graphe. Par conséquent, quelques travaux ont proposé des canonisations non exactes. Les expérimentations menées par [Kemmar *et al.*2013] ont illustré que le nombre des sous-graphes perdus ou en surplus est négligeable par rapport au gain en temps de calcul comparativement à des approches exactes. Mais malgré sa complexité, une représentation canonique est calculée qu'une seule fois, à la génération d'un nouveau candidat, ce qui motive la popularité de son utilisation.

3.3.2 Représentations canoniques

Généralement, un graphe est représenté comme une matrice d'adjacence ou une liste d'adjacence. La figure 3.8 illustre un exemple de la représentation d'un graphe (a) par une matrice d'adjacence (b) et par une liste d'adjacence (c).

La difficulté liée à ces représentations est qu'un même graphe peut avoir de nombreuses représentations en dépendance de l'ordre des sommets au niveau de la matrice ou la liste. La figure 3.9 illustre un exemple de deux graphes G et H isomorphe et leurs représentations matricielles correspondantes α et β respectivement. Nous remarquons facilement que les deux graphes G et H sont isomorphes simplement en appliquant les substitutions suivantes $v_0 = v_0, v_1 = v_3, v_2 = v_2, v_3 = v_1$.

Alors que les représentations matricielles ne sont pas identiques. Pour faire face à ce dernier problème sans avoir de recours à l'isomorphisme de graphes, il est nécessaire de représenter chaque graphe par un code unique invariable par rapport à l'ordre des

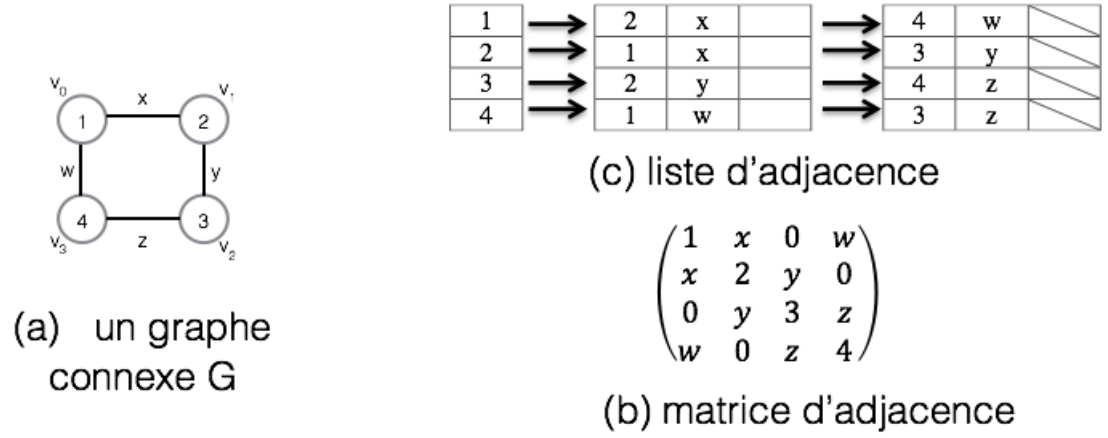

FIGURE 3.8 – La représentation des graphes.

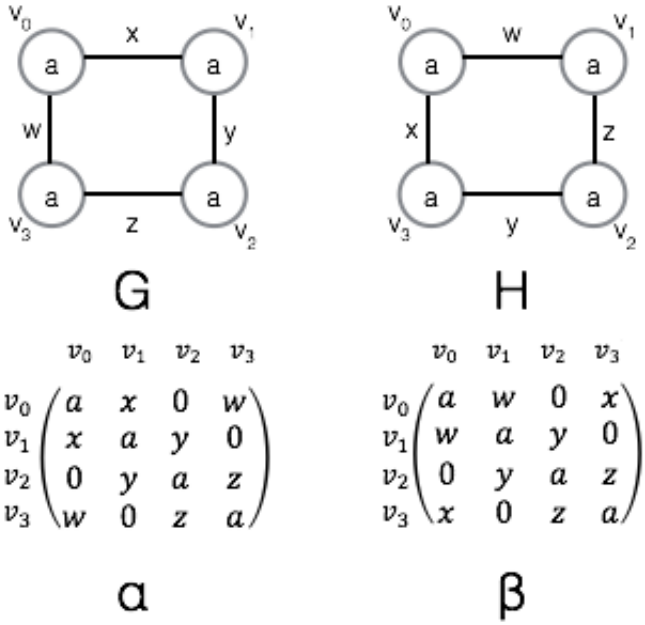

FIGURE 3.9 – Isomorphisme de graphes.

sommets et des arêtes. Par conséquent, deux graphes qui auront la même représentation canonique sont isomorphes. De nombreuses canonisations ont été présentées ces dernières années, soit pour le cas général, et même pour les cas particuliers p. ex. la concaténation des valeurs du triangle haut de la matrice d'adjacence (upper triangular matrix part) [Inokuchi *et al.*2000], DFS_{min} (depth first subscripting) [Yan and Han2002], BBP (block and bridge preserving) [Horváth *et al.*2010], ECE (edge and converse edge

triplets)[Thomas *et al.*2009],etc.

Triangle haut de la matrice d'adjacence

L'une des canonisations les plus populaires pour le cas des graphes représentés comme des matrices d'adjacences a été proposée par [Inokuchi *et al.*2000]. Cette canonisation est une chaine de caractères issue de la concaténation des étiquettes des sommets et des arêtes des colonnes du triangle haut de la matrice. Cette chaine de caractère peut être choisie comme le code lexicographique le plus spécifique (maximal) ou le plus général (minimal). Pour obtenir ce code canonique, les différentes permutations des sommets au niveau de la matrice vont être testées.

Par exemple par rapport aux graphes G et H (figure 3.9) le code canonique est aaaa zxyw. La chaine aaaa est obtenue par la canonisation des étiquettes des sommets et zxyw par la concaténation des étiquettes des arêtes issues des colonnes de la portion haute de la matrice (triangle haut). La configuration de la matrice d'adjacence utilisée pour obtenir ce code canonique, est illustrée sur la figure 3.10. Cette canonisation est très coûteuse avec une complexité d'ordre factoriel $\mathcal{O}(|V|!)$. Cette complexité peut être réduite considérablement en pratique, où plusieurs heuristiques ont été proposées afin d'élaguer les nombreuses permutations non intéressantes.

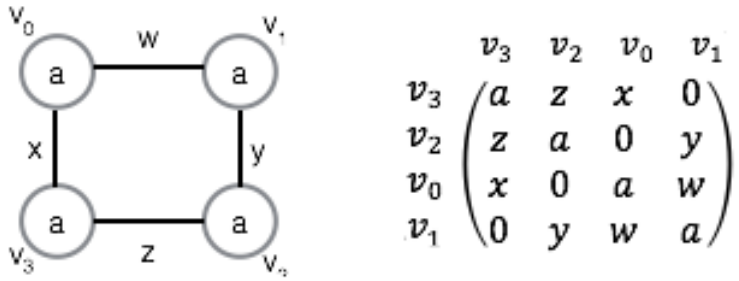

FIGURE 3.10 – La configuration de matrice d'adjacence canonique.

L'ordre lexicographique DFS

Proposé par [Yan and Han2002], cette canonisation est le résultat d'une exploration en profondeur d'un graphe connexe donné. D'abord le graphe G est exploré en commençant par un sommet donné, chaque sommet est sur étiqueté par rapport à l'ordre dans lequel il a été visité. Cette exploration en profondeur permet la construction d'un arbre noté G_T. Les arêtes contenues dans ce dernier arbre sont notées comme des arêtes avants E_f et celles qui n'existent pas sont notées comme arêtes arrières E_b. Ces arêtes avants et

arrières sont représentées comme des $5-tuples$ (i, j, et_i, et_e, et_j) où i, et j représentent l'ordre dans lesquels les deux sommets aux étiquettes et_i et et_j ont été visitées. et_e est l'étiquette de l'arête entre les deux sommets. Afin d'avoir la possibilité de décider si un code correspond à un graphe ou non, une relation d'ordre est associée à ces arêtes d'avant et d'arrière. Cette relation d'ordre $\prec$ est défini comme suit :

Soit $e_1 = (v_{i_1}, v_{j_1})$ et $e_2 = (v_{i_2}, v_{j_2})$

$$e_1 \prec e_2 = \begin{cases} \forall e_1, e_2 \in E_f \textit{ si et seulement si } j_1 < j_2 \\ \forall e_1, e_2 \in E_b \textit{ si et seulement si } i_1 < i_2 \text{ où } i_2 = i_2 \textit{ et } j_1 < j_2 \\ e_1 \in E_b, e_2 \in E_f \textit{ si et seulement si } i_1 < j_2 \\ e_1 \in E_f, e_2 \in E_b \textit{ si et seulement si } j_1 < i_2 \end{cases} \tag{3.1}$$

Comme pour le cas de canonisation précédente, de nombreux codes DFS peuvent être générés pour un même graphe. La figure 3.11 illustre sur la partie (b) un exemple de 4 arbres DFS possibles pour 4 parcours différents d'un même graphe G sur la partie (a). Sur la table 3.1 les 4 codes DFS sont présentés. Alors lequel est le code canonique parmi eux ? Le code canonique est le code le plus petit noté DFS_{min}. Un code DFS d'un graphe G est déclaré DFS_{min} s'il est le code d'ordre lexicographique le plus petit de tous les codes DFS de G. Par rapport à l'exemple de la figure 3.11 le code canonique de G est β. Le code β a été sélectionné comme le code canonique en étant le code le plus petit comparativement à tous les codes DFS possibles. La relation d'ordre entre les différents codes DFS d'un même graphe G est définie comme suit : soit $\prec_e$ une relation d'ordre entre les arêtes des différents codes, deux codes DFS α=$(a_0, a_1, ..., a_m)$ et β=$(b_0, b_1, ..., b_n)$, alors α est plus petit que β si et seulement si :

$$e_1 \prec e_2 = \begin{cases} \exists t, 0 \leqslant t \leqslant min(m, n), a_k = b_k \textit{ pour } k < t, a_t \prec_e b_t \\ \text{où: } a_k = b_k \textit{ pour } 0 \leq k \leq m \textit{ et } n \geq m \end{cases} \tag{3.2}$$

Comme nous pouvons facilement le constater, cette canonisation est aussi très coûteuse, parce qu'avant de pouvoir décider si un code est canonique tous les codes possibles devront être calculés, avec une complexité d'ordre factoriel $\mathcal{O}(|V|!)$.

Code d'arêtes et arêtes converses

Contrairement aux deux canonisations précédentes l'encodage d'arêtes et d'arêtes converses [Thomas *et al.*2009] est une canonisation dédiée à une classe particulière des graphes qui est celle des graphes à étiquetages d'arêtes non redondants. Un encodage d'un graphe G consiste à représenter par un ensemble composé des arêtes, arêtes converses, items des triplets connexes, et les items des formes. Comme instance les codes des graphes

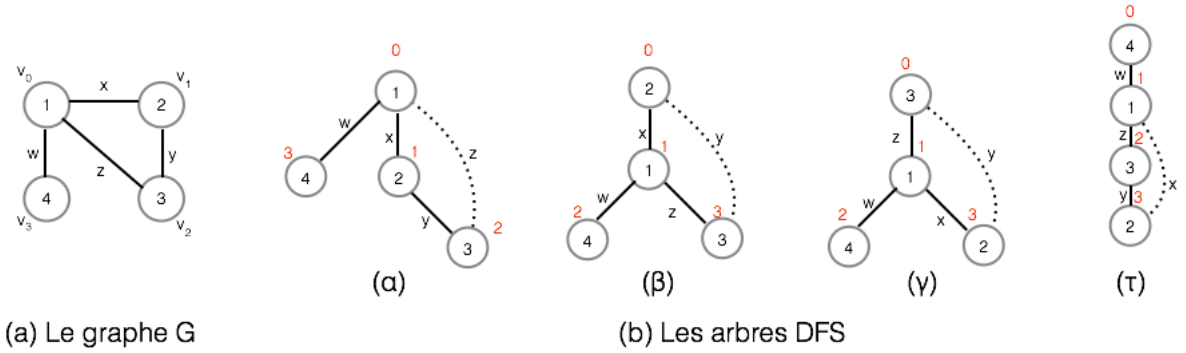

FIGURE 3.11 – Canonisation DFS.

TABLE 3.2 – Codes DFS

Les graphes	Les encodages
α	$(0,1,1,x,2),\ (1,2,2,y,3),\ (2,0,3,z,1),\ (0,3,1,w,4).$
β	$(0,1,2,x,1),\ (1,2,1,w,4),\ (1,3,1,z,3),\ (3,0,3,y,2).$
λ	$(0,1,3,z,1),\ (1,2,1,w,4),\ (1,2,1,x,3),\ (3,0,2,y,3).$
τ	$(0,1,4,w,1),\ (1,2,1,z,3),\ (3,2,2,y,3),\ (3,1,2,x,1).$

sur la Figure 3.12 sont illustrés sur le tableau 3.2.

Comme nous pouvons facilement le constater au niveau de la table 3.2, un code canonique d'un graphe peut être défini relativement à sa taille. S'il est inférieur à trois, le code canonique est soit le code de l'unique arête. S'il est de taille un (p. ex. le code du graphe (b)), et les codes des deux arêtes et de l'arête inverse pour un graphe de taille deux (p. ex. le code du graphe (c)). Alors que pour une taille supérieure ou égale à trois un code canonique est défini comme les codes de toutes les arêtes et les arêtes converses ainsi que les codes des triplets et ceux de formes de tous les sous-graphes connexes de tailles trois (p. ex. (d), (e), (f)). L'item utilisé pour la représentation de trois arêtes connexes est l'ensemble des étiquettes des trois arêtes précédées par "**".

Ces items de triplets permettent de distinguer le fait que les trois arêtes sont connexes, mais ils ne permettent pas de distinguer la forme que prend ce triplet d'arêtes connexes. Alors la question est : est-ce que ces triplets forment un triangle, un spike, ou une chaine ?

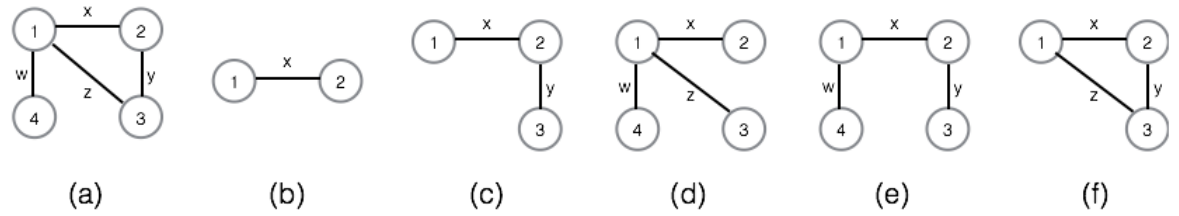

FIGURE 3.12 – Graphes à étiquetage d'arêtes non redondant.

TABLE 3.3 – Les encodages correspondants aux graphe sur la figure 3.12.

Les graphes	Les encodages
(a)	(1,x,2), (1,z,3), (1,w,4), (2,y,3), (w,1,x), (x,2,y), (x,1,z), (w,1,z), (y,3,z), (**,x,y,z), (**,w,x,z), (**,w,y,z), (**,w,x,y), (x,y,z,150), (w,x,z,200), (w,y,z,100), (w,x,y,100).
(b)	(1,x,2).
(c)	(1,x,2), (2,y,3), (x,2,y).
(d)	(1,x,2), (1,z,3), (1,w,4), (w,1,x), (w,1,z), (x,1,z), (**,w,x,z), (w,x,z,200).
(e)	(1,x,2), (1,w,4),(2,y,3),(w,1,x),(x,2,y),(**,w,x,y),(w,x,y,100).
(f)	(1,x,2), (1,z,3), (2,y,3), (x,2,y), (x,1,z), (y,3,z), (**,x,y,z), (x,y,z,150).

Pour distinguer la forme de chaque triplet au niveau du code canonique un item de forme est ajouté. Cet item est composé en plus des étiquettes des trois arêtes, d'un identifiant, qui sera noté par 100 pour une chaine, 200 pour un spike, et enfin 150 pour un triangle (p. ex. les codes des graphes (d), (e), (f)).

À partir de la définition ci-dessus, nous pouvons déduire qu'un encodage d'un graphe consiste en : 1) l'ensemble des unions des encodages de ses sous-graphes de taille 3 $\mathcal{E}$ si sa taille est supérieure ou égale à trois. Sinon c'est les encodages correspondants à ses arêtes et arêtes converses. La complexité de ce type d'encodage est de l'ordre polynomial $\mathcal{O}(|E|^3)$. En pratique cette complexité est fortement liée au nombre des sous-graphes de

taille 3. Pour pouvoir utiliser cet encodage pour le cas général, il faut faire attention aux symétries soit par rapport aux arêtes, ou les sous-graphes de taille trois. Ce processus augmente la complexité de l'encodage où la solution nécessite un temps exponentiel.

Enfin, dans la littérature, il existe beaucoup d'autres techniques de canonisation soit pour des classes de graphes particulières ou pour le cas général. Nous citons à titre d'exemple [Huan *et al.*2003], [Horváth *et al.*2010], etc.

3.3.3 La vérification de fréquence

Malgré la complexité de la phase de génération des motifs, la phase de vérification (c.-à-d. de calcul, et de validation du prédicat) de fréquence reste la phase clé pour un processus de découverte de motifs fréquents. Alors, pour un processus de découverte des sous-graphes connexes fréquents, le calcule de la fréquence consiste à tester l'existence d'un isomorphisme de sous-graphes de chacun des candidats qui ont été générés par rapport à chacun des graphes connexes de la base de graphes $G_i \in \mathcal{D}$.

Le problème d'isomorphisme de sous-graphes

Encore plus difficile que l'isomorphisme de graphes, l'isomorphisme de sous-graphes est connu comme étant un problème de complexité $NP-complet$ [Garey and Johnson1979]. Cette difficulté est due à la complexité de la structure manipulée, ce qui nécessite l'exploration des deux graphes pour vérifier les différentes substitutions des sommets et des arêtes possibles. Pour tester si un graphe G_1 de taille t_1 est un sous-graphe d'un graphe G_2 de taille $t_2 > t_1$, il faut tester l'isomorphisme de graphes entre G_1 et chacun des sous-graphes de G_2 de taille t_1. Si l'un des sous-graphes de G_2 est isomorphe à G_1 alors le test est validé.

La figure 3.13 illustre trois graphes X_1, X_2 et X_3 où X_1 est un sous-graphe isomorphe de X_2 dû au fait qu'il est isomorphe à g_1 l'un des sous-graphes de X_2 g_1, g_2 et g_3. Alors que X_1 n'est isomorphe à aucun des sous-graphes de X_3 (c.-à-d. g_4, g_5, g_6 et g_7) ce qui signifier que X_1 n'est pas un sous-graphe de X_3. Autrement dit, le graphe G_2 contient le graphe G_1, noté $G_1 \subset G_2$, s'il existe un isomorphisme de G_1 à un sous-graphe de G_2 (c.-à-d. une bijection entre G_2 et l'un des sous-graphes de G_1).

Contrairement au problème de redondance (c.-à-d. l'isomorphisme de graphes), les DSF (c.-à-d. les algorithmes de découverte des sous-graphes fréquents) n'ont pas eu beaucoup d'espace de manoeuvres. Beaucoup de techniques ont été proposées pour résoudre ce dernier problème avec un objectif en commun qui est de réduire autant que possible le temps de calcul, soit en utilisant une comparaison (c.-à-d. matching) exacte

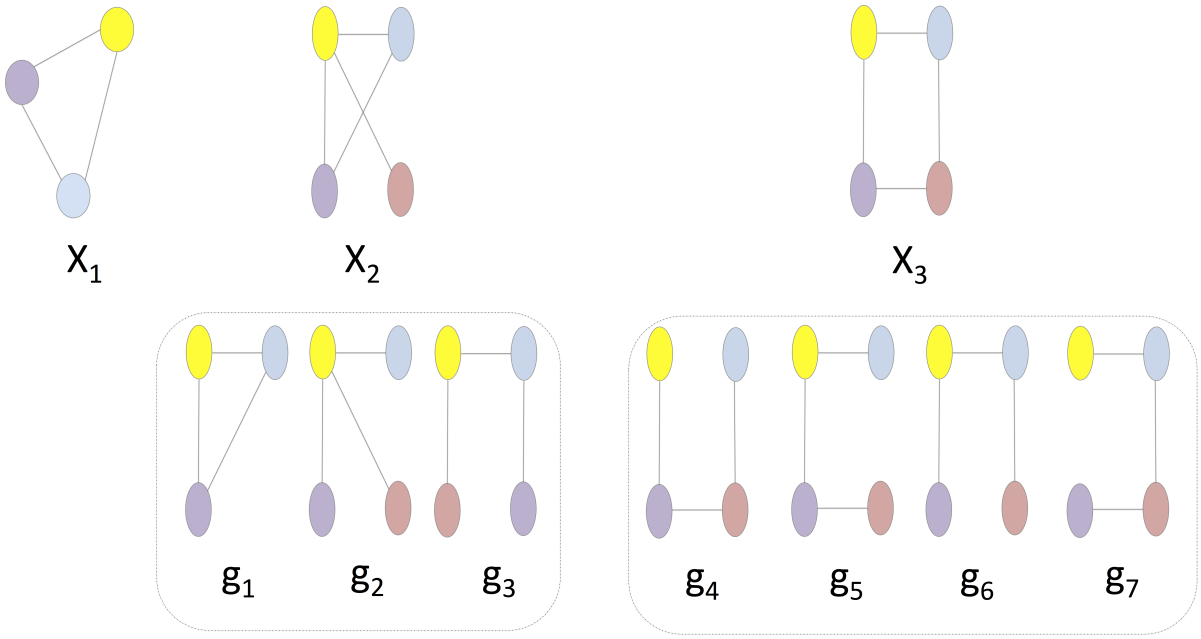

FIGURE 3.13 – Exemple d'isomorphisme de sous-graphes.

[Ullmann1976], [Schmidt and Druffel1976], [McKay1981] où en acceptant une certaine tolérance. Autrement dit, les deux orientations utilisent soit des techniques de test d'isomorphisme de sous-graphes exacts (NP-complet) ou des techniques approchées qui n'assurent dans aucun cas l'extraction de l'information complète. Dans ce manuscrit, nous considérons l'orientation vers la résolution d'isomorphisme de sous-graphes exact. Alors afin de contourner ce problème, les DSF s'engagent principalement à réduire le nombre de tests, en évitant de générer et de tester la fréquence des candidats qui n'ont aucune chance d'être fréquents et les candidats qui ont déjà été testés.

3.4 Quelques algorithmes de découverte des sous-graphes connexes fréquents

Comme nous l'avons déjà mentionné, beaucoup d'algorithmes ont été développés pour la découverte des sous-graphes connexes fréquents. Sur cette section, nous présentons trois algorithmes les plus populaires.

3.4.1 AGM

AGM (Apriori Graph Mining) [Inokuchi *et al.*2000] est l'un des premiers algorithmes développés pour la découverte de sous-graphes fréquents. AGM manipule des graphes représentés comme des matrices d'adjacences et propose une exploration en largeur de

l'espace de recherche. En suivant, le modèle d'APRIORI, AGM procède niveau par niveau, afin d'énumérer tous les sous-graphes fréquents. Ces sous-graphes sont induits, mais pas nécessairement connexes. Commençons avec le vide, tous les sous-graphes contenant un seul sommet sont générés, et à chacun des niveaux suivants un ensemble de sous-graphes candidats contenant un sommet de plus qu'au niveau précédent est généré à partir des sous-graphes fréquents de ce dernier niveau, en vérifiant la fermeture descendante de ces candidats. Le pseudocode de l'algorithme AGM est illustré sur l'algorithme 1.

Input: Un base de graphes $\mathcal{D}$, et un seuil minimum δ.
Output: Les sous-graphes fréquents $\mathcal{F}$.

$\mathcal{F} = \emptyset$;
$C_1 =$ tous les sommets étiquetés de $\mathcal{D}$;
$k = 0$;
while $C_{k+1} \neq \emptyset$ **do**
  $k = k + 1$;
  $\mathcal{F}_k = \emptyset$;
  for *chaque graphe* $H \in C_k$ **do**
    if H *est fréquent* **then**
      $\mathcal{F}_k = \mathcal{F}_k \cup H$
    end
  end
  $\mathcal{F} = \mathcal{F} \cup \mathcal{F}_k$;
  $C_{k+1} = jonctions(\mathcal{F}_k)$;
end
return $\mathcal{F}$

Algorithm 1: | AGM.

Contrairement au cas des itemsets, la génération des sous-graphes candidats, ainsi que le test de fréquence est largement plus complexe. Pour la génération des candidats la jonction de deux matrices d'adjacences est nécessaire, ainsi que la canonisation afin d'éviter la génération des redondants. Alors que pour la vérification de la fréquence des coûteux tests d'isomorphismes de sous-graphes sont utilisés.

AGM considère un graphe G à n sommets comme une matrice de dimension $n * n$ dont un élément diagonal a_{ii} est l'étiquète du sommet v_i et un élément non diagonal a_{ij} est l'étiquète de l'arête liant le sommet i au sommet j.

AGM génère chaque sous-graphe candidat du niveau $k+1$ sommets par la jonction de

deux sous-graphes fréquents de k sommets, qui ne diffèrent que d'un seul sommet. À titre d'exemple, soit les deux graphes G, H à k sommets dont les matrices d'adjacence sont α, et β :

$$\alpha = \begin{pmatrix} \alpha_{k-1} & \alpha_k \\ \alpha_k^T & \alpha_{kk} \end{pmatrix}$$
$$\beta = \begin{pmatrix} \beta_{k-1} & \beta_k \\ \beta_k^T & \beta_{kk} \end{pmatrix}$$

Ces deux matrices α, β sont identiques $\alpha_{k-1} = \beta_{k-1}$ à l'exception des éléments du $k-$ème ligne et colonne ($\alpha_k \neq \beta_k$, $\alpha_k^T \neq \beta_k^T$, $\alpha_{kk} \neq \beta_{kk}$), la jonction des matrices α et β en résulte la matrice :

$$\lambda = \begin{pmatrix} \alpha_{k-1} & \alpha_k & \beta_k \\ \alpha_k^T & \alpha_{kk} & l \\ \beta_k^T & l & \beta_{kk} \end{pmatrix}$$

Cette matrice λ est la représentation d'un graphe de $k+1$ sommet, où α, β sont inclus, mais le problème qui s'impose par rapport à cette situation est : est-ce que les deux sommets α_{kk} et β_{kk} sont liés où non ? Et s'ils le sont quelle est l'étiquette de cette arête ? Soit L_α et L_β les ensembles des étiquettes des arêtes de α et β, il y a $(|L_\alpha \cup L_\beta| + 1)$ valeurs possibles pour l'élément l. Par conséquent, l'union de deux matrices α et β entraînera la création de $(|L_\alpha \cup L_\beta| + 1)$ sous-graphes candidats différents. La figure 3.14 illustre la génération d'un sous-graphe candidat à 5 sommets à partir de deux sous-graphes de 4 sommets ayant a, et b comme étiquettes de sommets, et a, b, c, d comme étiquettes d'arêtes. Selon les étiquettes d'arêtes, nous pouvons ainsi créer 5 sous-graphes candidats différents. Un autre problème est aussi lié à cette opération de jonction de matrices, la redondance de génération d'un même graphe. Par ailleurs, la matrice d'adjacence, obtenue lors de l'opération de jonction dépend de l'ordre dans lequel les deux matrices sont considérées $\alpha \cup \beta \neq \beta \cup \alpha$. Alors, pour éviter la génération du même candidat plusieurs fois, les deux sous-graphes G, H sont joints dans l'ordre $G \cup H$ si et seulement si le code canonique de la matrice α de G est lexicographiquement inférieure à celui de la matrice β qui représente H $canon(\alpha) \preceq canon(\beta)$.

Comme nous l'avons déjà mentionné, après avoir généré un candidat sa fréquence doit être vérifiée. Le calcul de la fréquence d'un sous-graphe revient à un bon nombre de

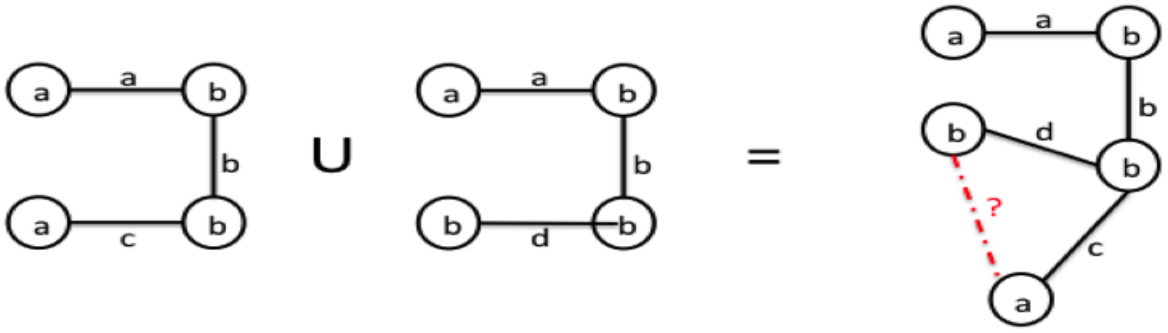

FIGURE 3.14 – Jonction à l'apriori.

tests d'isomorphisme de sous-graphes. AGM réutilise certains résultats de calculs achevés aux étapes précédentes afin d'accélérer le test d'isomorphisme de sous-graphes. Cette accélération consiste à élaguer l'arbre de recherche des substitutions possibles où chaque niveau représente l'association d'une paire de sommets.

3.4.2 gSpan

L'algorithme gSpan (Graph-based Substructure PAtterN mining) [Yan and Han2002], explore en profondeur l'espace de recherche. Un seul sous-graphe est traité à la fois, dont toutes les extensions sont explorées avant l'exploration d'un autre sous-graphe de la même taille. L'espace de recherche hiérarchique prend la forme d'un arbre, où chaque noeud correspond à un code DFS d'un graphe. Ces codes DFS sont des séquences qui permettent l'identification de deux graphes isomorphes, alors le processus de découverte des sous-graphes fréquents revient alors à une exploration d'un espace des séquences. À première vue l'exploration est très coûteuse, car chaque graphe peut avoir plusieurs codes DFS correspondants sur l'espace de recherche, mais si cette exploration est faite selon l'ordre lexicographique des séquences (l'ordre dans lequel le code DFS est construit), tous les sous-arbres ayant comme racines des codes qui ne sont pas des codes canoniques peuvent être élagués, puisque les noeuds de codes canoniques correspondent à des graphes isomorphes qui ont été déjà explorés.

Comme la stratégie de recherche adoptée par gSpan traite un seul sous-graphe à la fois. Alors à partir d'un sous-graphe fréquent une extension fréquente qui n'a pas déjà été visitée parmi toutes les extensions possibles de ce sous-graphe est générée. gSpan étend un sous-graphe fréquent en lui ajoutant soit a) une arête en reliant deux sommets du graphe, ou b) bien un nouveau sommet ainsi qu'une arête le reliant à un sommet du graphe. Cependant cette tâche d'extension doit faire attention à ce que l'ajout de cette

nouvelle arête génère encore un code DFS (c.-à-d. une nouvelle arête, ne peut pas être ajoutée n'importe où). Les arêtes avant, qu'elles soient reliées à de nouveaux sommets ou reliées à des sommets déjà existants, elles ne peuvent pas être ajoutées que sur le chemin extrémal de l'arbre DFS. Par contre les arêtes arrière ne peuvent pas être ajoutées que pour relier deux sommets déjà existants et plus précisément elles ne peuvent pas relier que le sommet maximal à un autre sommet du chemin extrémal.

Input: Un base de graphes $\mathcal{D}$, et un seuil minimum δ.
Output: Les sous-graphes fréquents $\mathcal{F}$.

main $(\mathcal{D}, \delta)$;
$\mathcal{F} = \emptyset$;
$V_{\mathcal{I}}$= tous les sommets non fréquents de $\mathcal{D}$;
$E_{\mathcal{I}}$= tous les arêtes non fréquents de $\mathcal{D}$;
$V_{\mathcal{F}}$= tous les sommets fréquents de $\mathcal{D}$;
$E_{\mathcal{F}}$= tous les arêtes fréquents de $\mathcal{D}$;
for *chaque graphe* $G_i \in \mathcal{D}$ **do**
  retirer les sommets V_I et les arêtes E_I infréquents;
end
$S_{\preceq}$ = trier $E_{\mathcal{F}}$ en ordre croissant de code;
for *chaque arête* $e \in S_{\preceq}$ **do**
  générer un graphe g ne contenant que e;
  $\mathcal{F} = \mathcal{F} \cup$découverte$(\mathcal{D}_e, g, \delta)$;
  for *chaque graphe* $G_i \in \mathcal{D}$ **do**
    retirer l'arête e;
  end
end
return $\mathcal{F}$ *;*

Algorithm 2: | gSpan.

La figure 3.15 illustre un exemple d'extension de deux arbres DFS. Le chemin en arêtes rouges et sommets gris représente le chemin extrémal de l'arbre DFS. Sur l'arbre (a), l'ajout d'une arête avant est réalisable seulement à partir de l'un des sommets 1,2,3 vers le nouveau sommet bleu. Pour l'arbre (b), l'ajout d'une arête arrière n'est pas réalisable que sur le sommet extrémal et l'un des sommets sur le chemin extrémal, pour cet exemple du sommet extrémal 2 vers le sommet 4. La restriction des possibilités d'ajout d'arêtes a comme objectif de construire un code DFS valide après l'ajout d'une nouvelle arête. Cela peut être facilement visualisé sur l'exemple de la figure 3.15. L'ajout d'une arête avant sur l'arbre DFS (a) est toujours vers la fin du code alors le sommet bleu ajouté aura une étiquette comme j étant visité maximale (p. ex. 5) alors cette arête e (i,j,e) est

Input: Un base de graphes $\mathcal{D}$, un graphe G et un seuil minimum δ.
Output: Les sous-graphes descendants de G.

$Des = \emptyset$;
if *code(G)≠canon(G)* **then**
  return
else
  $Des = Des \cup G$;
  Sc = tous les sous graphes issue de l'extension de G par une arête;
  for *chaque candidat* $g \in Sc$ **do**
    if g *est fréquent* **then**
      $Des = Des \cup$ découverte$(\mathcal{D}, g, \delta)$)
    end
  end
end
return Des

Algorithm 3: La procédure découverte ().

d'ordre lexicographique supérieur (plus spécifique) par rapport à n'importe quelle arête du code. De même, pour le choix du sommet extrémal, pour faire des arêtes arrière, d'une part, la surétiquette associée à ce sommet est maximale alors l'ajout d'une arête arrière à partir de ce sommet est surement d'ordre lexicographique le plus spécifique. D'autre part, l'ajout du sommet extrémal vers un sommet du chemin extrémal a comme but de garder l'ordre du code après le retour arrière. À titre d'exemple, sur la figure 3.15 (a), une arête arrière à partir du sommet 3 vers le sommet 4 est impossible parce que cela violera l'ordre linéaire du code DFS.

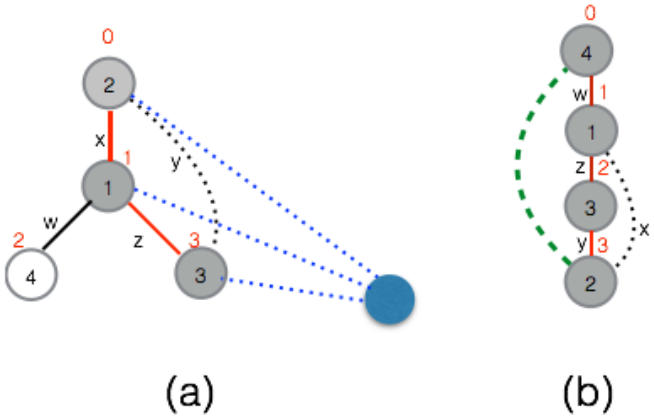

FIGURE 3.15 – Les extensions des codes DFS.

En plus que les contraintes limitent les possibilités d'extensions, la stratégie de recherche menée par gSpan s'appuie au fait que toute extension d'un sous-graphe non fréquent ne peut pas être fréquente. En raison de la fermeture descendante, alors les extensions de sous-graphes non fréquents peuvent être évitées sans risquer de manquer des sous-graphes fréquents, ce qui permet d'élaguer implicitement un grand nombre de sous-graphes non fréquents. Par conséquent une réduction importante de l'espace de recherche, ce qui permet automatiquement la réduction du nombre d'opérations de génération ainsi que du calcul de fréquence des candidats. D'autres optimisations ont été mises en oeuvre par gSpan pour la réduction des tests de fréquences où : 1) l'ensemble des arêtes non fréquentes E_I et les arêtes déjà traitées de tous les graphes de $\mathcal{D}$ sont retirés, sans risque d'élimination d'un sous-graphe fréquent. 2) Pour chaque sous-graphe $g_1(V_{g_1}, E_{g_1})$ traité, la fréquence ne sera pas testée que pour le sous-ensemble de graphes $\mathcal{D}_g = \{G \in \mathcal{D} \ | E_{g_1} \subseteq E_G\}$.

L'algorithme 2 illustre le pseudocode de gSpan. D'abord, l'algorithme identifie les ensembles de sommets fréquents $V_{\mathcal{F}}$ et les arêtes fréquentes $E_{\mathcal{F}}$, ainsi que les sommets et les arêtes non fréquentes $V_{\mathcal{I}}$, $E_{\mathcal{I}}$ dans $\mathcal{D}$. Les sommets et les arêtes non fréquents sont retirés. Ensuite, l'ensemble des arêtes fréquentes est trié en suivant un ordre lexicographique croissant des codes. Par la suite, chaque arête $e \in E_{\mathcal{F}}$ est explorée sur le sous-ensemble de graphes $\mathcal{D}_e$ de $\mathcal{D}$ avec la procédure découverte(). Cette dernière procédure retourne l'ensemble des sous-graphes fréquents qui sont les descendants de cette arête e (c.-à-d. un graphe composé d'une seule arête). Ces sous-graphes fréquents sont alors ajoutés à l'ensemble des sous-graphes fréquents $\mathcal{F}$. À partir du fait que tous les sous graphes fréquents qui contiennent e ont été récupérés cette dernière peut être retirée de tous les graphes de $\mathcal{D}$. Pour chaque sous-graphe candidat G la procédure découverte() s'exécute d'une manière récursive comme suit : d'abord, le code du candidat est testé si c'est bien un DFS_{min}, sinon il est inutile de l'explorer, car un graphe isomorphe à ce candidat a déjà été exploré. Si le code est bien canonique, le sous-graphe candidat est énuméré comme étant un sous-graphe descendant fréquent. Ensuite, tous les sous-graphes successeurs immédiats de G par l'ajout d'une arête sont générés et leurs fréquences sont testées. Trié en suivant l'ordre lexicographique croissant de leur code, les sous-graphes successeurs immédiats fréquents de G sont explorés par la procédure découverte().

3.4.3 Gaston

Gaston (GrAph/Sequence/Tree extractiON) [Nijssen and Kok2005], est un autre algorithme de découverte des sous-graphes connexes fréquents. Cet algorithme propose une recherche adaptée par rapport à trois niveaux de complexités (les chemins, les arbres et

les graphes). Motivé par beaucoup de résultats expérimentaux qui indiquent que la plupart des sous-structures fréquentes récupérées par rapport à une base de graphes sont des arbres [Nijssen and Kok2005] [Huan *et al.*2004]. Encore, les chemins et les arbres sont largement plus simples à gérer par rapport aux graphes. Par conséquent, Gaston propose d'investir la découverte en trois phases consécutives : la découverte des chemins, les arbres, et enfin les graphes connexes fréquents respectivement. En effet, Gaston est développé afin de se comporter comme un algorithme spécialisé de découverte : de chemins fréquents face aux chemins, de sous-arbres fréquents face aux arbres, et enfin des sous-graphes connexes fréquents face aux graphes.

Input: Un base de graphes $\mathcal{D}$, et un seuil minimum δ.
Output: Les sous-graphes fréquents $\mathcal{F}$.

$\mathcal{F} = \emptyset$;
$V_{\mathcal{F}}$= tous les sommets fréquents de $\mathcal{D}$;
$E_{\mathcal{F}}$= tous les arêtes fréquents de $\mathcal{D}$;
$\mathcal{PATH}_{\mathcal{F}} = recup - chemins(V_{\mathcal{F}},\ E_{\mathcal{F}},\ \mathcal{D},\ \delta)$;
$\mathcal{A}_{\mathcal{F}} = recup - arbres(\mathcal{PATH}_{\mathcal{F}},\ V_{\mathcal{F}},\ E_{\mathcal{F}},\ \mathcal{D},\ \delta)$;
$\mathcal{G}_{\mathcal{F}} = recup - arbres(\mathcal{PATH}_{\mathcal{F}} \cup \mathcal{A}_{\mathcal{F}},\ E_{\mathcal{F}},\ \mathcal{D},\ \delta)$;
$\mathcal{F}$=$\mathcal{P}_{\mathcal{F}} \cup \mathcal{A}_{\mathcal{F}} \cup \mathcal{G}_{\mathcal{F}}$;
return $\mathcal{F}$;

Algorithm 4: | Gaston.

Pour atteindre l'objectif déclaré, cet algorithme commence par l'énumération des chemins. Ensuite, il énumère les arbres connexes et enfin les sous-graphes connexes. Lors des phases d'énumération de chemins et des arbres, Gaston réduit les augmentations possibles à celles qui ne permettent que la génération des chemins à partir d'autres chemins, et d'éviter la génération des cycles pour le cas des arbres. L'algorithme 4 illustre le pseudocode de Gaston. D'abord, l'algorithme identifie les ensembles des sommets fréquents $V_{\mathcal{F}}$ et des arêtes fréquentes $E_{\mathcal{F}}$. Ensuite, il récupère l'ensemble des chemins $\mathcal{PATH}_{\mathcal{F}}$ fréquents. Par la suite, à partir des chemins récupérés sur la phase précédente, l'ensemble des arbres $\mathcal{A}_{\mathcal{F}}$ fréquents est récupéré. Enfin, à partir de l'ensemble des chemins et des arbres récupérés, les graphes connexes $\mathcal{G}$ sont générés pour récupérer que les fréquents $\mathcal{G}_{\mathcal{F}}$ entre eux.

La génération des candidats

Comme nous l'avons déjà mentionné, la génération des candidats consiste à produire de nouveaux motifs qui n'ont pas été déjà générés à partir d'un ou plusieurs motifs qui ont déjà été visités. Pour Gaston, la génération d'un nouveau candidat (c.-à-d. chemin,

arbre, ou graphe) consiste en une opération d'augmentation en ajoutant une arête à la fois.

La génération des chemins

La principale difficulté liée à la génération de nouveaux chemins à partir d'autres est qu'un chemin admet deux orientations. À titre d'exemple, soit le chemin (e) sur la figure 3.12, il peut être encodé comme 4w1x2y3 ou 3y2x1w4. En plus, chaque chemin admet potentiellement deux parents (prédécesseurs) immédiats, par la suppression de l'un de ses noeuds (sommets) d'extrémités. Pour éviter la régénération d'un même chemin deux fois par augmentation à partir de ses deux parents, Gaston considère comme un unique prédécesseur immédiat que le plus petit entre les deux, et le code canonique du chemin est augmenté à partir de ce dernier.

Encore, consacrer la première phase de l'algorithme pour l'énumération que des chemins permet de limiter le nombre d'augmentations possibles. D'abord par l'élimination des arêtes qui lient des noeuds déjà existants (génèrent des cycles). Ensuite, chacune des arêtes qui permettent de générer un chemin qui admet pour prédécesseur immédiat canonique, un chemin autre que celui utilisé pour l'augmenter. Alors en général la génération de nouveaux chemins est définie comme suit : si le chemin est symétrique alors toutes les augmentations possibles sont appliquées à l'une de ses extrémités. Autrement, pour un chemin non symétrique les augmentations sont appliquées par rapport aux deux extrémités. Les chemins générés sont maintenus, que si le chemin original avec lequel ils ont été générés est leur prédécesseur immédiat canonique.

La génération des arbres

Après l'énumération des chemins fréquents $\mathcal{PATH}_{\mathcal{F}}$, la seconde phase consiste à générer des arbres en commençant par ces chemins fréquents. Pour l'énumération des arbres, la stratégie adoptée par Gaston est basée sur des techniques développés pour des arbres racinés [Nijssen and Kok2003]. Pour ces arbres racinés, la génération d'un nouveau candidat par augmentation d'une arête est beaucoup plus simple. Cette augmentation consiste en l'attachement d'un nouveau noeud à un noeud du chemin de l'extrême droite (c.-à-d. rightmost path) de l'arbre. Par conséquent, le nombre de possibilités d'augmentation est considérablement réduit.

Pour investir ces principes d'énumération des arbres simples n'ayant pas de racines, en considérant un code du chemin prédécesseur (c.-à-d. backbone string) pour chacun des arbres simples, l'espace de recherche est décomposé en classes d'équivalences. Chacune des

classes regroupe un ensemble d'arbres simples qui partagent le même code du chemin prédécesseur. Pour chaque arbre simple, un code du chemin prédécesseur est calculé, comme suit : D'abord, l'ensemble P_{max} de ses chemins maximaux est récupéré. À partir d'un chemin d'entre eux le (bi)-centre est distingué. Ce (bi)-centre est la racine de l'arbre simple. Si le chemin est d'une taille impaire, alors le noeud au centre du chemin ($\frac{|P_{max}|+1}{2}$) est considéré comme la racine de l'arbre. Pour une taille paire du chemin, deux noeuds sont marqués comme racines. Ces deux noeuds sont aux positions $i,\ et\ i+1$ tel que $i = \frac{|P_{max}|}{2}$. En commençant, de la racine (c.-à-d. le centre de chacun des chemins maximaux P_{max}), tous les chemins maximaux sont encodés comme des chaines de caractères des étiquettes des sommets et d'arêtes. Pour le cas de deux racines (bi-centre) le code d'ordre lexicographique le plus petit de chacune des racines est récupéré. Pour le cas d'un centre unique, les deux codes d'ordre lexicographique le plus petit sont récupérés. Ensuite, pour les deux cas, les deux codes récupérés sont concaténés comme suit : le code de l'ordre le plus petit entre les deux est renversé, puis concaténé au deuxième code d'ordre le plus grand.

Pour la génération des nouveaux candidats à partir d'un code d'arbre, Gaston suit le procédé suivant : D'abord, le lien entre les deux racines est supprimé pour le cas de bi-centre, et pour le cas d'un centre unique, ce centre est supprimé. Chacun des deux sous-codes résultats est considéré comme un arbre raciné, et il peut être augmenté au niveau de son chemin d'extrême droite (c.-à-d. rightmost path). Ensuite, les deux sous codes sont regroupés pour former un nouvel arbre simple. Finalement, pour garantir la non-redondance des candidats, la génération d'un nouvel arbre simple n'est pas valide que si le code du chemin prédécesseur du nouvel arbre ne change pas par rapport à l'arbre original.

La génération des graphes

Après avoir énuméré l'ensemble des chemins et des arbres simples fréquents, au niveau de cette troisième étape, toutes les augmentations qui permettent de générer des cycles, qui ont été maintenues par rapport aux deux étapes précédentes seront appliquées. Alors pour l'encodage des graphes, il est composé de deux séquences qui se succèdent. La première séquence consiste au code de l'arbre simple original (chemin original codé comme un arbre), et la seconde consiste en l'ensemble d'augmentations appliquées au code sur la première séquence. Les augmentations au niveau de cette deuxième séquence sont codées comme des tuples de la forme (v_i, v_j, l) pour v_i et v_j deux sommets d'ordre lexicographique d'étiquettes de sommets correspond à $v_i < v_j$ et l l'étiquette de l'arête entre eux.

Pour un graphe donné, son code canonique est celui d'ordre lexicographique le plus

petit entre tous ses codes possibles. Alors, pour la génération d'un nouveau graphe par augmentation, un nouveau tuple est ajouté à la fin du code canonique de l'un de ses prédécesseurs immédiats, et pour que ce nouveau code soit canonique (valide) si et seulement si le tuple à ajouter est d'ordre lexicographique plus grand que du dernier déjà ajouté. Autre que cette dernière condition qui permet la réduction de nombreuses possibilités d'augmentations qui mènent vers des codes non canoniques, ce procédé permet de garantir que les graphes ne sont pas augmentés qu'à partir de leurs arbres couvrants.

Calculer la fréquence

Comme nous l'avons déjà mentionné, comme tout algorithme de découverte des sous-graphes connexes fréquents, calculer la fréquence est la phase la plus complexe. Gaston adopte une stratégie d'exploration en profondeur, et investit la possibilité de stocker des informations supplémentaires. Alors pour chaque sous-graphe énuméré, la liste des plongements de ce dernier par rapport à l'ensemble des graphes de la base de données est maintenue. Ce procédé permet de restreindre le nombre de graphes de la base de données pour lesquels un nouveau graphe candidat sera testé, s'il est sous-graphe isomorphe ? Donc ces informations supplémentaires permettent d'accélérer le calcul de la fréquence. Mais en contrepartie, elles nécessitent beaucoup d'espace mémoire pour sauvegarder les traces de chacun des graphes (chemins, arbres, et graphes) énumérés.

Ces listes de plongements associés à chaque graphe énuméré consistent en des tuples représentés comme suit : 1) pour un sommet isolé x, chacun des tuples représente un graphe G_i dans la base de données et un sommet y isomorphe à x (c.-à-d. G_i, y). 2) Pour un graphe augmenté au niveau des deux premières phases (c.-à-d. chemin, ou arbre), chaque tuple est composé de l'indice du plongement précédent, ainsi que le graphe dans la base de données et le nouveau sommet ajouté lors de l'opération d'augmentation. 3) Pour un graphe augmenté par renfermement de cycle, la liste consiste en des pointeurs vers ses parents (ses prédécesseurs immédiats). À partir d'une liste de plongements d'un graphe donné, sa fréquence est égale au nombre des différents graphes.

3.5 La découverte des sous graphes fréquents fermés et maximaux

Un défi majeur lié à la découverte des motifs fréquents est la présence d'un grand nombre de motifs fréquents. Par conséquent l'énumération de l'intégralité des motifs, ainsi que les analyser est quasi impossible, étant donné que tous les sous-motifs (généralisations) d'un motif fréquent donné X sont par la suite fréquents. À titre d'exemple,soit

un ensemble fréquent X de plusieurs éléments, alors, les 2^n sous-ensembles de X sont aussi fréquents. Ce phénomène est en raison de la propriété de fermeture descendante. Pour appréhender ce grand nombre de motifs, la fouille des sous-motifs fréquents fermés et maximaux est adoptée étant donné que l'ensemble des motifs fréquents fermés et maximaux est largement plus petit que celui de l'ensemble complet des motifs fréquents.

La complexité du problème de découverte des motifs fréquents fermés et maximaux est liée conformément à la complexité des motifs à chercher. En plus de l'influence vis-à-vis de la génération des candidats et le test de fréquence, cette complexité des motifs détermine aussi la forme (les propriétés) de l'espace de recherche ($\mathcal{L}$). Cet espace de recherche joue un rôle majeur afin d'éviter la génération, ainsi que le test de tous les motifs fréquents avant d'atteindre tous les maximaux ou les fermés. Ce qui permet par conséquent d'avoir un délai polynomial pour l'énumération de deux motifs fermés ou maximaux successifs, si le test de fréquence est polynomial.

Par exemple pour les problèmes où cet espace de recherche prend la forme d'un treillis ou un produit de chaines, ce qui n'est pas le cas des graphes. L'algorithme "dualize-and-advance" permet de résoudre le problème de découverte des maximaux, efficacement et même dans un délai polynomial si et seulement si le test de fréquence est polynomial [Gunopulos *et al.*1997], ce qui est le cas des ensembles, des itemsets, des clés [Gunopulos *et al.*2003], et des séquences rigides [Nourine and Petit2012]. Tandis qu'à l'égard de motifs fréquents fermés, l'algorithme "divide-and-conquer" [Boley *et al.*2010] permet de résoudre ce dernier problème efficacement en visitant juste un nombre équivalent au cardinal des fermés. Néanmoins, pour pouvoir utiliser cet algorithme, l'espace de recherche doit être fortement accessible. Semblablement au problème des maximaux, les motifs fermés sont énumérables dans un délai polynomial si et seulement si le test de fréquence est polynomial, ce qui est encore le cas des ensembles, des itemsets, et des séquences.

Pourtant, bien que l'espace de recherche des sous-graphes connexes d'un graphe est fortement accessible, le délai d'énumération ne peut pas être polynomial tant que l'isomorphisme des sous-graphes ne l'est pas. L'application de "divide-and-conquer" permet juste la minimisation des sous-graphes parcourus avant d'atteindre les motifs fermés. Par conséquent, l'énumération des sous-graphes fermés peut être résolue en parcourant au plus $|\mathcal{C}|$ sous-graphes (c.-à-d. la génération de $|\mathcal{C}|$ graphes inductifs générateurs $g' = g \cup e$ où g est un graphe fermé), et en testant l'isomorphisme des sous-graphes (c.-à-d. $\subseteq_{sgi}$) $|\mathcal{C}|$ fois. Ce qui est nettement plus avantageux que l'énumération de tous les sous-graphes fréquents $\mathcal{F}$,

pour enfin atteindre les fermés entre eux. Ce qui nécessite de visiter$|\mathcal{F} \cup min(\overline{\mathcal{F}})|$ ($min(\overline{\mathcal{F}})$ est l'ensemble des sous-graphes minimaux non fréquents) sous-graphes et de tester pour chacun d'entre eux l'isomorphisme des sous-graphes. Comme pour le cas des fermés, l'énumération des sous-graphes maximaux souffre des mêmes problèmes. Le meilleur algorithme Margin [Thomas *et al.*2010], il a été développé afin de parcourir que l'ensemble des sous-graphes autour de la bordure qui sont notés promoteurs $\widehat{\mathcal{F}}$. Margin visite $|\widehat{\mathcal{F}}|$ sous-graphes et exerce $|\widehat{\mathcal{F}} \cup min(\overline{\mathcal{F}})|$ tests d'isomorphisme de sous-graphes.

Dans la littérature, il n'y a pas de nombreux algorithmes développés pour faire face au nombre exponentiel de sous-graphes connexes fréquents qui forment un espace de recherche $\mathcal{L}$, et qui permettent atteindre très rapidement les sous-graphes fréquents fermés et maximaux. À notre connaissance, cela est dû à la difficulté de l'espace de recherche et le coût exclusif du test de fréquence. Ces deux problèmes limitent l'espace de manoeuvre pour proposer des solutions optimales, contrairement aux problèmes où l'espace de recherche forme un treillis ou plus au moins fermé sous intersection.

L'avantage de l'extraction que des motifs fermés (ou maximaux) a comme but de : (1) réduire l'espace de recherche pour compresser le nombre d'opérations élémentaires (test de fréquence et génération de candidats). (2) limiter le nombre des résultats pour relaxer le processus d'analyse par la suite.

Alors, le fait d'extraire ou de parcourir tous les sous-graphes fréquents $\mathcal{F}$ pour enfin trouver les fermés ou les maximaux parmi eux, ce qui va être sans intérêt par rapport à l'optimisation de l'espace de recherche et même très coûteux, où le nombre de sous-graphes visités ainsi que les tests d'isomorphisme sous-graphes nécessaires sera $|\mathcal{F} \cup min(\overline{\mathcal{F}})|$. Afin d'éviter de parcourir tous les sous-graphes fréquents $\mathcal{F}$, les algorithmes de découverte des sous-graphes fréquents fermés et maximaux existants utilisent plusieurs techniques d'élagage afin de réduire l'espace de recherche $\mathcal{L}$. Dans cette section, nous discutions les quatre algorithmes les plus populaires qui ont été proposés pour résoudre ce problème : CloseGraph (extraction des sous-graphes connexes fermés fréquents) [Yan and Han2003] ; SPIN, et Margin (extraction des sous-graphes connexes maximaux fréquents)[Huan *et al.*2004], [Thomas *et al.*2010] ; et ISG (extraction des sous-graphes connexes maximaux fréquents à étiquetage d'arêtes non redondant)[Thomas *et al.*2009].

3.5.1 Closegraph

L'algorithme le plus connu pour la découverte des sous-graphes connexes fréquents fermés est CloseGraph, cet algorithme se comporte de manière récursive en utilisant

une exploration en profondeur de l'espace de recherche et un prolongement à l'extrême droite pour la génération de nouveaux candidats. Cet algorithme est basé sur gSpan [Yan and Han2002], avec l'addition d'une technique d'élagage afin d'éviter la génération de tous les sous-graphes fréquents. Cette technique d'élagage permet l'arrêt anticipé du processus de recherche pour certaines branches pour lesquelles la condition de terminaison précoce est satisfaite. Cette idée est fondue sur le fait qu'un sous-graphe g' contenant g pour chacune de ses instances dans la base de données $\mathcal{D}$, alors il sera plus intéressent de faire des extensions directement à partir de g' que de les faire à partir de g. Cela est dû à ce que g n'est pas fermé et qu'il est peut probable qu'il existe un sous-graphe fermé qui ne peut être généré qu'à partir de g.

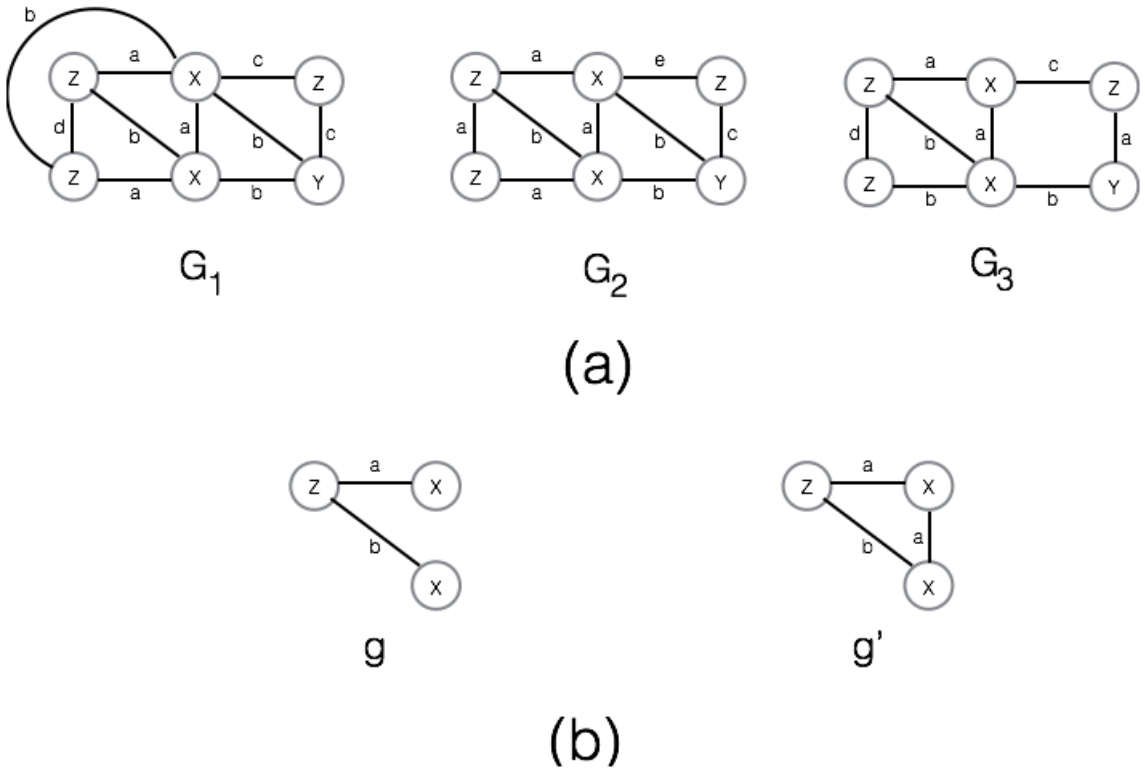

FIGURE 3.16 – L'équivalence d'occurrences.

Autrement dit, cette condition d'élagage est basée sur une équivalence d'occurrences d'un sous-graphe g et l'un de ses successeurs immédiats $g' = g \cup e$ ce qui permet de décider que les descendants du graphe g ne contenant pas l'arête e ne seront fort probablement pas des fermés. La figure 3.16 illustre un exemple de condition de terminaison précoce. Soit la base de graphes $\mathcal{D}$ sur la partie (a) et les deux graphes g et g' sur la partie (b). Le nombre d'occurrences de g dans $\mathcal{D}$ est : $\vartheta(g, \mathcal{D}) = 2 + 1 + 0 = 3$ et g' dans $\mathcal{D}$ est :

$\vartheta(g', \mathcal{D}) = 2+1+0 = 3$, ainsi que le nombre d'occurrences de g' contenant g dans $\mathcal{D}$ est : $\iota(g, g', \mathcal{D}) = 2+1+0 = 3$. Le fait que $\vartheta(g, \mathcal{D}) = \iota(g, g', \mathcal{D})$ l'arbre de recherche est élagué au niveau de g.

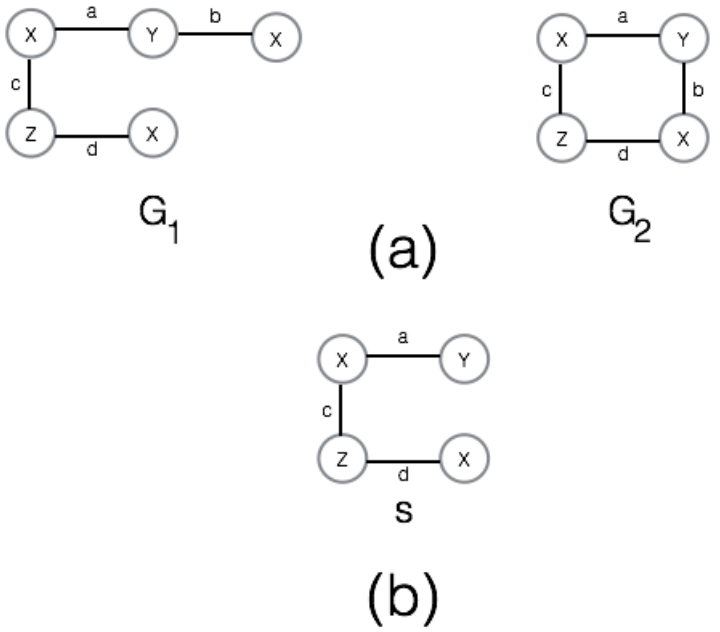

FIGURE 3.17 – La faille de la terminaison précoce.

La vérification de la validité de cette condition de terminaison précoce est généralement très coûteuse et n'offre pas une certitude de complétude des sorties. En respectant le principe d'équivalence d'occurrences, plusieurs sous-graphes fermés peuvent être manqués. Un temps de calcul plus considérable doit être consommé afin de récupérer ces sous-graphes manqués. Les expérimentations réalisées ont montré que la détection de ces cas d'échec nécessite la moitié du temps total nécessaire. La figure 3.17 illustre un exemple d'une configuration (c.-à-d. une base de graphes sur la partie (a)) pour laquelle un sous-graphe fréquent peut être oublié en élaguant l'espace de recherche, le graphe S sur la partie (b) de la figure. Considérons le sous-graphe $g =$ $\overset{a}{(x)—(y)}$ et le successeur immédiat $g' =$ $\overset{a}{(x)—(y)}\overset{b}{—(x)}$. Le nombre d'occurrences de g : $\vartheta(g, \mathcal{D}) = 2$ et le nombre d'occurrences de g' contenant g : $\iota(g, g', \mathcal{D}) = 2$, donc en respectant le principe de la terminaison précoce, nous allons arrêter les extensions (la recherche) à partir de g, en faveur de celles de g' parce que toute instance de g est une instance de g'. Malheureusement, nous remarquons que le sous-graphe S sur la partie (b) de la figure est un graphe fréquent

fermé qui ne peut être atteint qu'à partir de g.

Alors pour résoudre ce cas de faille, une détection passive est utilisée, l'idée se concrétise comme suit : d'abord, il est supposé qu'il n'existe aucune faille. Alors le processus de recherche est exécuté normalement jusqu'à ce qu'une extension double avec une même arête qui forme deux sous-graphes non fréquents sera possible c.-à-d. une arête donnée e peut faire objet d'une extension avant, et arrière (en ajoutant un nouveau sommet ou en reliant deux sommets existants). Alors la condition de terminaison précoce sera cassée, et les extensions à partir du sous-graphe g seront exécutés.

3.5.2 Spin

Spin [Huan *et al.*2004] est le premier algorithme qui a été développé pour la découverte des sous-graphes connexes maximaux fréquents. Cet algorithme s'exécute principalement sur deux étapes, premièrement tous les arbres fréquents $\mathcal{F}_t$ sont extraits. Deuxièmement, des groupes de sous-graphes fréquents sont construits à partir de $\mathcal{F}_t$, afin que chaque groupe de sous-graphes fréquents forme une classe d'équivalence. Les motivations pour lesquelles Spin maintient ces deux étapes sont : les opérations sur les arbres comme la normalisation, l'isomorphisme d'arbres et de sous-arbres sont de loin plus simple que celles employées sur les graphes. Les expérimentations sur des données synthétiques ainsi que sur plusieurs domaines d'applications ont démontré que les arbres représentent la fraction majoritaire de l'ensemble de tous les sous-graphes fréquents.

En résumé, par rapport à ces motivations, nous remarquons facilement que Spin investit le fait que généralement la plupart des sous-graphes fréquents ayant été extraits par un algorithme de découverte sont réellement des arbres. En contrepartie aux nombreux avantages offerts par ce choix d'extraction d'arbres, ce grand nombre d'arbres explorés avant d'arriver aux graphes nécessite un temps de traitement important, sachant que pour des bordures de séparation plus élevées (seuil de fréquence très bas) l'ensemble des maximaux sera majoritairement composé de sous-graphes (arbres cycliques). Encore aussi, l'utilisation de ce nombre important de sous-arbres pour enfin construire pour chaque sous-arbre entre eux un groupe de sous-graphes fréquents, afin que chaque groupe de sous-graphes fréquents forme une classe d'équivalence. Chaque classe d'équivalence est composée de sous-graphes qui partagent le même arbre couvrant (Spanning tree) canonique $t \in \mathcal{F}_t$.

Spin définit l'arbre couvrant canonique d'un graphe donné g comme étant l'arbre couvrant (c.-à-d. un arbre contenant tous les sommets de g) maximal parmi tous les arbres couvrants possibles en respectant un certain ordre total défini entre les codes BFCS

(Breadth-First Canonical String) de ces arbres après une phase de normalisation. La figure 3.18 illustre un exemple d'un graphe G sur la partie haute gauche et l'ensemble de ses sous-arbres couvrants. L'arbre T_1 est le sous-arbre couvrant maximal de G c.-à-d. canonique.

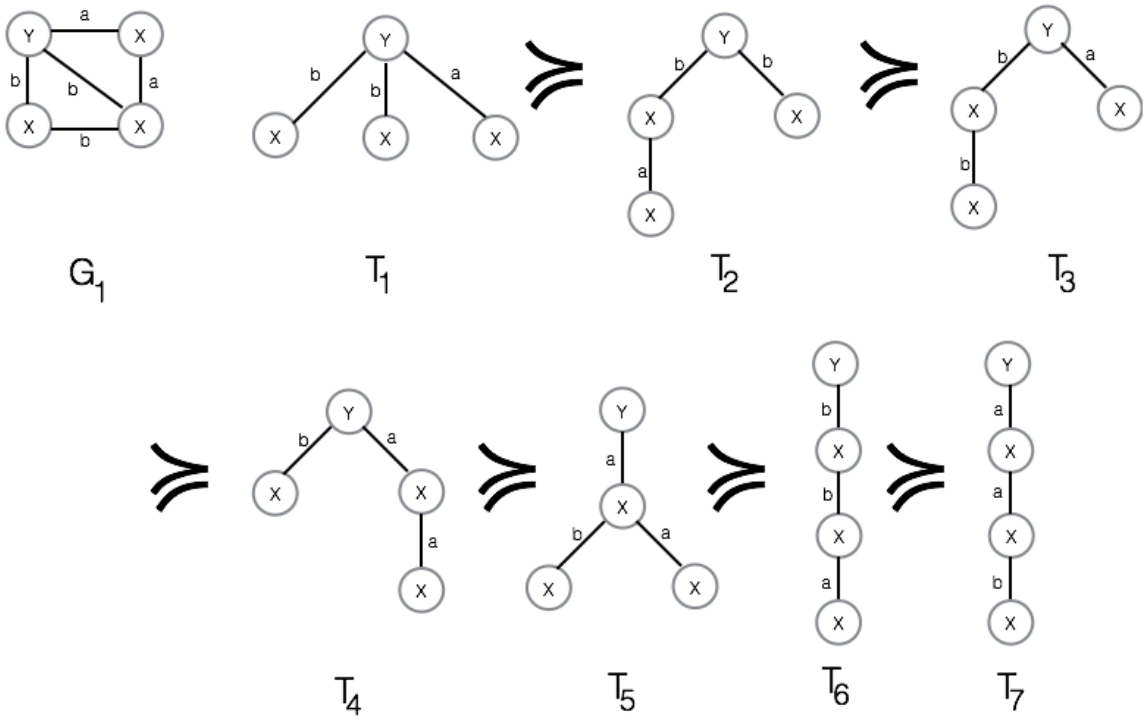

FIGURE 3.18 – Exemple de sélection d'un arbre couvrant canonique.

En considérant chaque arbre $t \in \mathcal{F}_t$ extrait à partir de la première phase comme étant un arbre couvrant canonique, Spin s'engage à l'énumération des sous-graphes fréquents maximaux à partir de l'ensemble des sous-graphes ayant t comme un sous-arbre couvrant canonique. Cependant, cette dernière étape n'est pas efficace parce que lors de l'énumération de tous les sous-graphes fréquents en les construisant et testant leurs fréquences, cela reviendra à énumérer le tous c.-à-d. $|\mathcal{F}|$. Pour éviter ce problème, quelques techniques d'optimisation sont intégrées (Buttom-Up Pruning, Tail Shrink, and External-Edge Pruning) afin d'accélérer le processus de parcours et d'éviter de parcourir tous les sous-graphes dans les classes d'équivalences. Cet ensemble de techniques d'élagage vise à supprimer des classes totalement ou partiellement dans le but de trouver rapidement les sous-graphes fréquents maximaux.

3.5.3 Margin

Margin [Thomas *et al.*2010] est un autre algorithme qui vise la découverte des sous-graphes maximaux fréquents, motivé par l'idée que généralement, les sous-graphes fréquents maximaux se situent au centre de l'espace de recherche, ce qui rend la tâche de les récupérer très coûteuse en temps de calcul, soit par un algorithme naïf qui parcoure le tous $|\mathcal{F}|$ avant d'atteindre les maximaux, soit par Spin qui va énumérer au moins tous les arbres fréquents $\mathcal{F}_t$ avant d'atteindre les maximaux. Pour surmonter ce problème, les auteurs de Margin proposent d'éviter l'exploration de l'espace de recherche au-dessus et en dessous de la bordure (c.-à-d. la frontière entre les motifs qui sont considérés fréquents et ceux qui ne sont pas) en se limitant à parcourir que les sous-graphes qui se trouvent autour de cette bordure (noté sous-graphes prometteurs $\widehat{\mathcal{F}}$). Chaque élément $x \in \widehat{\mathcal{F}}$ est un sous-graphe fréquent immédiat d'un graphe non fréquent $y \notin \mathcal{F}$ c.-à-d. $\widehat{\mathcal{F}} = \{x \in \mathcal{F} | \exists y \notin \mathcal{F} \; et \; x \subset y \; et \; |y| = |x| + 1\}$. Le pseudocode sur l'algorithme 5 illustre les grandes lignes de Margin.

Input: Une base de graphes $D = \{G_1, ..., G_n\}$.
Output: L'ensemble de sous-graphes fréquents maximaux MF.

$MF = \emptyset$;
for $G_i \in \mathcal{D}$ **do**
    $LF = \varphi$;
    Récupérer le sous graphe représentative R_i de G_i ;
    ExpandCut$(LF, CR_i|R_i)$ ou CR_i est un enfant non fréquent de R_i
    Merge(MF, LF);
end
return MF

Algorithm 5: Margin.

D'abord, pour chaque graphe $G_i \in \mathcal{D} = \{G_1, ..., G_n\}$, Margin parcoure en profondeur l'espace de recherche $\mathcal{L}_i$ des sous-graphes connexes du graphe G_i, le sous-graphe représentatif $R_i \in G_i$ est récupéré. Ensuite, ExpandCut est invoquée avec comme première coupe R_i et l'un de ces fils non fréquents CR_i, tous les candidats $\widehat{\mathcal{F}}$ local LF sont récupérés. Pour chaque G_i les LF sont comparés avec les globaux MF déjà trouvés. Les sous-graphes maximaux fréquents globaux MF sont récupérés. La Figure 3.19 illustre l'espace de recherche exploré par Margin.

Afin d'atteindre son objectif, Margin d'abord explore l'espace de recherche en profondeur, soit en ajoutant un ensemble d'arêtes E une par une en commençant du vide $\emptyset$, ou

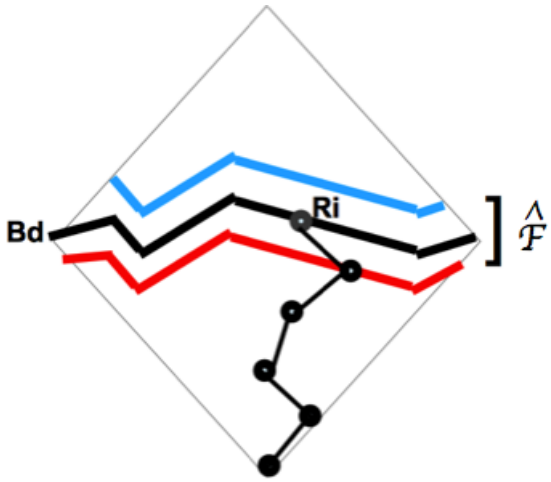

FIGURE 3.19 – L'espace de recherche considéré par Margin.

en supprimant un ensemble d'arêtes du graphe G_i jusqu'à ce qu'un sous-graphe $x \in \widehat{\mathcal{F}}$ (noté le sous-graphe représentatif R_i) sera trouvé. La deuxième étape de l'algorithme consiste à trouver récursivement tous les éléments de $\widehat{\mathcal{F}}$ en utilisant *ExpandCut* qui commence par une première coupe $(C|R_i)$ où C est un super-graphe non fréquent de R_i. La fonction *ExpandCut* saute d'un candidat à un autre jusqu'à ce que tous les sous-graphes de $\widehat{\mathcal{F}}$ seront trouvés.

Soit une coupe $(S|X)$ *ExpandCut* sera invoquée pour de nouvelles coupes en respectant les trois étapes suivantes :

Étape 1. Soit P_1, P_2 deux autres parents fréquents de S et P_3 non fréquent, à partir d'une coupe $cut(S|X)$ les deux coupes $cut(P_1|S)$ et $cut(P_2|S)$ sont invoquées et P_1, P_2 sont rapportés comme des éléments de $\widehat{\mathcal{F}}$. La figure 3.20 illustre un exemple du cas présent.

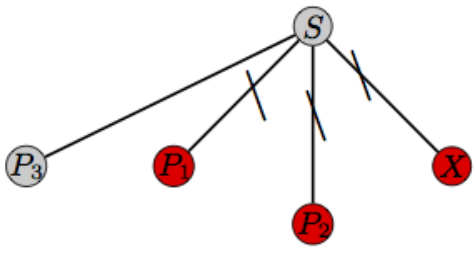

FIGURE 3.20 – Exemple de l'étape 1 de *ExpandCut*.

Étape 2. Pour une coupe $cut(S|X)$ initiale, toutes les coupes entre les sous-graphes parents fréquents P_f de S et leurs fils non fréquents C_f sont invoquées $cut(C_f|P_f)$,

les P_f sont énumérés comme des éléments de $\widehat{\mathcal{F}}$. Pour un fils C_2 non fréquent, et un autre fréquent C_1 de X en respectant la propriété de losange il existe un fils en commun $M = C_1 \cup C_2$ (le système est confluant), a cette situation la coupe $cut(C_1|M)$ est invoquée et C_1 est énuméré comme un élément de $\widehat{\mathcal{F}}$. La figure 3.21 illustre les deux cas cités dans l'étape 2.

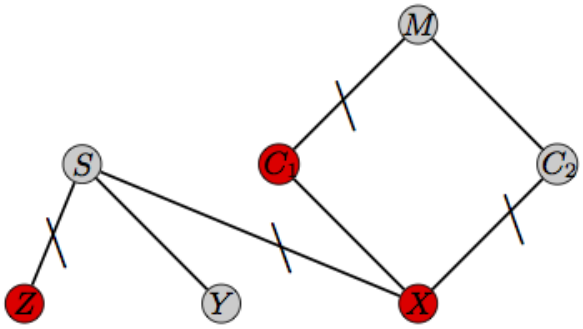

FIGURE 3.21 – Exemple de l'étape 2 de *ExpandCut*.

Étape 3. Pour une coupe $cut(S|X)$ pour un sous graphe parent non fréquent V de S une nouvelle coupe est invoquée pour l'un de ces parents fréquents W ($cut(V|W)$). Dans la figure 3.22 un exemple est illustré et V est un parent non fréquent de S alors la coupe est invoquée pour son parent fréquent W.

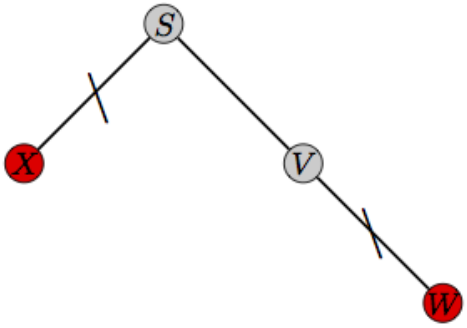

FIGURE 3.22 – Exemple de l'étape 3 de *ExpandCut*.

Enfin, les sous-graphes maximaux fréquents $\mathcal{M}_i$ sont extraits à partir de $\widehat{\mathcal{F}}$. Les expérimentations ont illustré que Margin est très performant avec des bases de graphes denses, mais ce n'est pas le cas par rapport à une base de graphes sparses. L'analyse du comportement de Margin nous amène à déduire que la complexité de Margin est liée au nombre des sous-graphes promoteurs $\widehat{\mathcal{F}}$ où chaque sous-graphe entre eux est obtenu après l'invocation de *ExpandCut* à partir d'un autre sous-graphes dans $\widehat{\mathcal{F}}$. Alors pour les

graphes sparses la bordure est généralement très basse ce qui est équivalant à dire que l'ensemble des promoteurs $\widehat{\mathcal{F}} = \mathcal{F} \cup Min(\mathcal{F})$. Par conséquent, Margin se comporte comme un algorithme naïf.

3.5.4 ISG

ISG [Thomas *et al.*2009] est un autre algorithme développé pour résoudre le problème de découverte des sous-graphes fréquents maximaux, mais contrairement aux algorithmes Margin et Spin, ce dernier ne supporte qu'une classe particulière des graphes qui sont les graphes à étiquetage d'arêtes non redondant. L'innovation apportée par ce dernier algorithme est la réduction du problème à une découverte d'itemsets maximaux fréquents. Cette réduction s'est concrétisée en encodant la base de graphes comme une base d'itemsets. Ensuite, un algorithme de découverte d'itemsets est appliqué afin de récupérer l'ensemble d'itemsets maximaux fréquents. Pour enfin décoder ces itemsets maximaux aux sous-graphes correspondants. Cependant, cet encodage n'assure pas la complétude des résultats [Krishna *et al.*2011].

Nous remarquons que le principal défi d'ISG est comment encoder des graphes connexes comme des itemsets ? et même le défi devient plus important face au décodage. Comme nous l'avons déjà mentionné au niveau de la sous-section 3.3.2, l'encodage dénoté arêtes et arêtes converses (edges and converse edges ECE) permet d'encoder un graphe comme un ensemble d'arêtes, d'arêtes converses, les items d'identification de chaque triplet d'arêtes connexes et les items d'identification de forme (type) des triplets. Cet encodage est d'une complexité d'ordre polynomial $\mathcal{O}(|E(G)|^3)$. L'algorithme 6 illustre le pseudocode d'un algorithme naïf qui permet l'encodage d'un graphe à étiquetages d'arêtes non redondants comme un ensemble d'items.

Malheureusement, malgré les nombreux avantages de la découverte des itemsets fréquents, les principales difficultés liées ISG sont au niveau de la phase de décodage. Ces difficultés sont principalement des itemsets correspondent :1) à des graphes non connexes , 2) à des graphes ambigus. Un code d'arêtes et d'arêtes converses $ECE(G)$ qui contient un item d'un triplet connexe sans la présence de son item de forme (type) est appelé un code ambigu. La figure 3.23 illustre un exemple d'un graphe non connexe (a) correspondant à l'encodage α sur la table 3.3. Pour pouvoir s'en apercevoir de cette déconnexion au niveau du code il suffit de vérifier qu'aucun groupe de triplets n'est isolé. Au niveau du code α, l'intersection d'items de triplets d'arêtes connexes (**,a,b,c) $\cap$ (**,d,e,f)=$\emptyset$. Le code β correspondent à un encodage ambigu, il est facile de distinguer que pour l'item du triplet $(**, a, b, c)$ il n'existe pas un item de type (c.-à-d. forme (a,b,c, ?)) correspondent. Pour le

Input: Un graphe avec un étiquetage d'arêtes non redondants $G(V, E)$.
Output: Un ensemble d'items $ECE(G)$.

for $e_1 \in E(G)$ **do**
  $e = (et_{v'}, et_{e_1}, et_{v''})$; // $et_{v'}, et_{v''}$ en ordre croissant des étiquettes $\kappa = \kappa \cup e$;
end
$ECE(G) = \kappa$;
for $e_1 \in E(G)$ **do**
  for $e_2 \in E(G) \backslash e_1$ **do**
    if $connexe(e_1, e_2)$ **then**
      $doub = (et', et_v, et'')$; // et', et'' en ordre croissant des étiquettes.
      $ECE(G) = ECE(G) \cup doub$;
    end
  end
end
for $e_1 \in E(G)$ **do**
  for $e_2 \in E(G) \backslash e_1$ **do**
    for $e_3 \in E(G) \backslash \{e_1, e_2\}$ **do**
      if $connexe(e_1, e_2, e_3)$ **then**
        $trip = (**, et, et', et'')$; // en ordre croissant des étiquettes.
        $ECE(G) = ECE(G) \cup trip$;
        $type = selectiontype(e_1, e_2, e_3)$;
        $trip' = (et, et', et'', type)$; // en ordre croissant des étiquettes.
        $ECE(G) = ECE(G) \cup trip'$;
      end
    end
  end
end
return $ECE(G)$

Algorithm 6: L'encodage ECE.

présent cas, nous ne pouvons pas décider la forme du triplet, le graphe correspondant à cet encodage β peut être l'un des trois graphes (b),(c),(d) sur la figure 3.23. Les encodages γ, λ, ω correspondent aux graphes (b),(c),(d) respectivement.

En conclusion, le schématique suivi par ISG afin d'énumérer l'ensemble de sous-graphes connexes fréquents maximaux à étiquetage non redondants d'arêtes peut se résumer comme suit : ISG encode la base de graphes comme une base transactionnelle d'itemsets $\mathcal{D}'$ en premier lieu. Ensuite, il applique un algorithme de la découverte d'itemsets maximaux. Pour chaque maximal d'entre eux un test d'ambiguïté est invoqué. Les codes non ambigus sont décodés aux graphes connexes correspondants et ceux qui sont ambigus vont subir une phase de prétraitement avant de les décoder. Le prétraitement consiste à

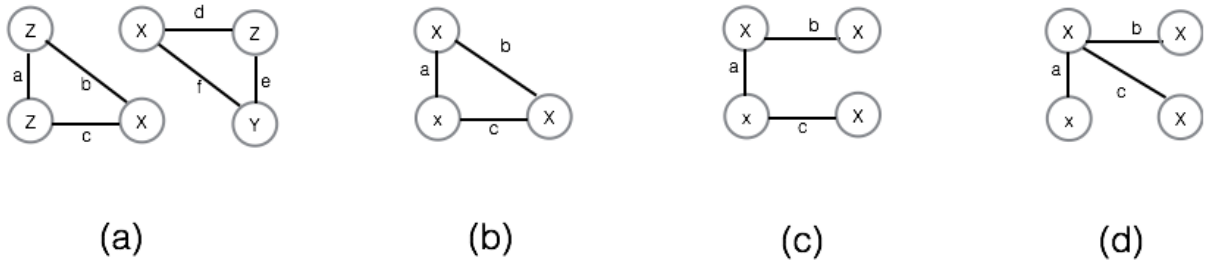

FIGURE 3.23 – Exemple de graphe non connexe et ambigu.

TABLE 3.4 – Les encodages correspondants aux graphe sur la figure 3.23.

Les graphes	Les encodages
α	(z,a,z), (x,b,z), (x,c,z), (a,z,b), (b,x,c), (a,z,c), (**,a,b,c), (a,b,c,150), (x,d,z), (y,e,z), (x,f,y), (d,z,e), (d,x,f), (e,y,f), (**,d,e,f),(d,e,f,150)
β	(x,a,x), (x,b,x), (x,c,x), (a,x,b), (b,x,c), (a,x,c), (**,a,b,c).
γ	(x,a,x), (x,b,x), (x,c,x), (a,x,b), (b,x,c), (a,x,c), (**,a,b,c), (a,b,c,150).
λ	(x,a,x), (x,b,x), (x,c,x), (a,x,b), (a,x,c), (**,a,b,c),(a,b,c,100).
ω	(x,a,x), (x,b,x), (x,c,x), (a,x,b), (b,x,c), (a,x,c), (**,a,b,c), (a,b,c,200).

détecter les triplets sans item de forme et éliminer l'ambiguïté en remplaçant les triplets en question par les couples d'arêtes qu'elles contiennent, ce processus peut générer jusqu'à trois sous-codes pour chaque triplet ambigu. Enfin une phase d'élagage est appliquée ce qui permettra l'élimination de tous les sous-graphes non maximaux.

3.6 La découverte d'un modèle à base de contraintes : la construction d'arbres de décisions

Comme nous l'avons mentionné précédemment, la constriction d'arbres de décisions est une autre spécialisation de la fouille des données à base de contraintes (Constraint-Based Mining). Pour construire un arbre de décision, nous nous sommes confrontés à de nombreuses difficultés liées soit : 1) à la sélection d'un bon échantillon de données d'ap-

prentissage pour faire pousser l'arbre, et un deuxième échantillon de test pour valider le modèle construit ; 2) au critère de sélection utilisé pour affecter les bons attributs aux bons noeuds en suivant un certain ordre de l'arbre ; 3) à la phase d'élagage qui est une étape clé dans le processus de construction d'un arbre de décision, afin de contourner les problèmes de sur apprentissage et de complexité issue de la phase d'augmentation.

Alors pour la construction d'un arbre de décision optimal pour une base de données donnée, nous devons faire de nombreux choix qui vont être décisifs pour la fiabilité des performances de l'arbre de décision construit. Ces choix commençant par la sélection d'un bon échantillon d'apprentissage, pour éviter l'utilisation des données erronées qui auront de mauvaises conséquences lors de la construction d'un arbre de décision ayant de bonnes performances par rapport aux exemples d'apprentissages, mais ne peuvent pas présenter autant de performances par rapport à un nouvel échantillon d'exemples. Encore, le choix de restreindre l'ensemble des attributs de la base de données à l'un de ses sous-ensembles d'attributs permet de construire pour chaque sélection un arbre de décision différent, et la meilleure sélection (c.-à-d. le choix du sous-ensemble d'attributs) est celle qui permet de construire l'arbre de décision ayant les meilleures performances. La figure 3.24 illustre un ensemble d'exemples sur un tableau de m attributs et n instances, en sélectionnant à chaque fois un sous-ensemble différent d'exemples d'apprentissage en utilisant seulement un sous-ensemble distinct d'attributs, un arbre de décision t_i est construit.

	at_1	at_2	at_3	at_{x-1}	at_x	at_{x+1}	at_{m-1}	at_m	classe
i_1	a1	b8	g5	d1	e3	f5	g5	h2	C_1
i_2	a2	b7	g6	d0	e2	f4	g4	h1	C_1
i_3	a3	b6	g7	d1	e1	f3	g3	h0	C_2
i_4	a4	b5	g8	d2	e0	f2	g2	h1	C_1
i_5	a5	b4	g9	d3	e1	f1	g1	h2	C_1
i_{x-1}	a6	b3	g0	d4	e2	f0	g0	h3	C_3
i_x	a7	b2	g1	d5	e3	f1	g1	h4	C_1
i_{x+1}	a8	b1	g2	d6	e4	f2	g2	h5	C_2
i_{n-1}	a9	b0	g3	d7	e5	f3	g3	h6	C_3
i_n	a10	b1	g4	d8	e6	f4	g4	h7	C_3

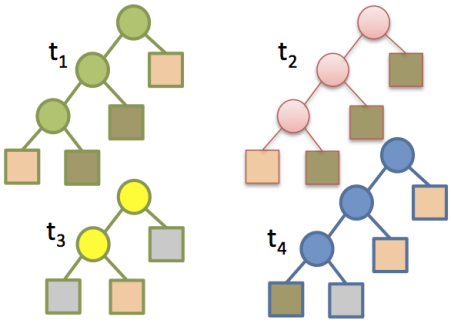

FIGURE 3.24 – Exemple de construction d'arbres de décisions.

Autre que la sélection des données, le choix de critère de sélection d'attributs est aussi un problème difficile dû au fait qu'il existe de nombreux critères de sélections (p. ex. Gini index, impurity-based criteria, twoing criterion) et pour un même échantillon de données d'apprentissage, de différents arbres de décisions sont construits en appliquant de différents critères de sélections. Alors pour faire un choix optimal, nous sommes dans

l'obligation d'étudier en détail toutes ces techniques afin de décider, laquelle d'entre elles qui est la plus adaptée pour un échantillon de données d'apprentissage donné. Plus encore, après que l'arbre a été poussé, il souffre généralement d'un problème de sur apprentissage, alors une phase d'élagage étant nécessaire, mais la difficulté par rapport à cette étape est qu'en plus du fait que les techniques existantes [Breiman *et al.*1984], [Quinlan1987], [Quinlan1993], [Luo *et al.*2013], souffrent de problèmes de sur élagage, ainsi que de sous élagage, la multitude des choix nécessite une étude approfondie de ces derniers afin de choisir laquelle d'entre elles sera la plus adaptée, et encore plus, les différentes expérimentations menées [Esposito *et al.*1997], [Mingers1987], [Bradford *et al.*1998], n'ont pas illustrées dans aucun cas une technique qui surclasse toutes les autres.

À partir de toutes ces difficultés, nous pouvons conclure que la construction d'un arbre de décision optimal est un problème de découverte d'un arbre de décision spécifique parmi tous les arbres de décisions possibles. Même en considérant que le choix de critère de sélection d'attributs comme étant lié au choix de modèle utilisé (p. ex. Cart [Breiman *et al.*1984], REPTree [Quinlan1987], J48 [Quinlan1993], etc.), ce dernier est bien aussi un problème complexe [Karabadji *et al.*2012a], [Karabadji *et al.*2012b], le choix de l'échantillon de données d'apprentissage (c.-à-d. les instances et les attributs utilisés) représente en lui même un problème de fouille des données à base de contraintes, où la question est de trouver un couple composé d'un sous-ensemble d'exemples et un sous-ensemble d'attributs qui permet de construire l'arbre de décision le plus performant pour un modèle donné. De même pour la phase d'élagage qui consiste à parcourir l'arbre poussé T_0 en remplaçant des sous-arbres comme des feuilles (c.-à-d. classes), un remplacement n'est valide que s'il permet d'amélioré la performance. Alors parmi tous ces remplacements possibles qui est l'arbre le plus performant parmi eux pour un modèle donné ?.

Conclusion du chapitre

Ces dernières années, beaucoup de méthodes de la fouille de données (données transactionnelles...) comme le clustering, la classification et la découverte des motifs fréquents... ont été étendues aux données structurées comme des graphes. Dans ce chapitre, nous nous sommes intéressés à la découverte des sous-graphes fréquents, où beaucoup de méthodes ont été présentées. Nous avons présenté aussi le processus de construction d'arbres de décisions et les difficultés liées à ce dernier, ainsi que quelques techniques d'élagage d'arbres de décisions. Enfin, nous avons illustré que les objectifs que ces méthodes d'élagages souhaitent atteindre peuvent être atteints simplement en utilisant d'autres alternatives. Sur les deux chapitres suivants, nous allons présenter nos contributions qui consistent en :

a) deux techniques pour la découverte des sous-graphes connexes à étiquetage d'arêtes non redondant. b) un algorithme qui permet de construire des arbres de décisions robustes sans faire appel à une phase d'élagage. Cet algorithme résout les problèmes de construction d'arbre de décision en cherchant une combinaison optimale d'un sous-ensemble d'exemples d'apprentissages, et un sous-ensemble d'attributs, qui permet de construire un arbre de décision non élagué très performant par rapport à des ensembles d'exemples indépendants de tests.

Deuxième partie

Contributions

Chapitre 4

La découverte des sous-graphes fréquents fermés à étiquetage d'arêtes non redondant

Sommaire

DANS ce chapitre, nous nous intéressons à l'énumération des sous-graphes connexes fréquents fermés pour la classe des graphes à étiquetage d'arêtes non redondant. Dans ce but, nous présentons les différentes difficultés liées à la complexité structurelle des graphes.

Dans un premier temps, nous présentons l'impact de la représentation des graphes et nous optons pour deux représentations : a) ensemblistes, et b) tableaux d'adjacences. Ces

deux représentations vont permettre d'illustrer la complexité des différentes opérations élémentaires sur les graphes. Cet impact est fortement remarquable, soit pour les principales opérations qui sont le test d'isomorphisme de graphes et de sous-graphes, soit pour les opérations : d'augmentation (ajouter une arête), ou de tester la connexité des graphes ayant été augmentés.

Dans un second temps, nous présentons un codage des graphes à étiquetages d'arêtes non redondants comme un ensemble d'items noté ECE (Edges and Converses Edges encoding [Thomas *et al.*2009]), où les opérations de test d'équivalence, et d'inclusion des ensembles d'items sont largement plus simples. Le problème avec cet encodage est qu'il n'est pas bijectif, où contrairement au fait que deux graphes isomorphes auront le même code (c.-à-d. un ensemble d'items), il existe une faille par rapport à la relation d'inclusion, alors cette dernière ne vérifie pas l'isomorphisme de sous-graphes dans tous les cas. Cette faille consiste à ce qu'un code d'un graphe g_1 qui n'est pas un sous-graphe de g_2 peut être inclus au niveau du code du graphe g_2. Dans le même contexte, nous proposons un test supplémentaire qui permet de résoudre cette faille et ainsi profiter au maximum de l'encodage ECE soit pour tester l'isomorphisme de graphes et même pour celui de sous-graphes. Plus encore, nous allons effectuer quelques tests comparatifs afin d'illustrer l'impact du test ajouté par rapport au temps total du test.

En troisième lieu, nous étudions les propriétés de l'espace de recherche, où contrairement aux cas de motifs moins complexes comme les itemsets, les clés, etc., pour lesquels l'aspect structurel des espaces de recherches permet de réduire considérablement le nombre des motifs à visiter (c.-à-d. à générer, à tester leurs fréquences, et qu'ils n'ont pas été visités auparavant), l'espace de recherche des graphes connexes est beaucoup plus complexe et ne permet pas l'adoption des mêmes manoeuvres ayant déjà été appliquées. Pour ce dernier problème, nous allons étudier cet espace de recherche et nous montrons qui il est bien fortement accessible. Dans ce contexte, nous investissons la forte accessibilité de l'espace de recherche des graphes connexes afin de minimiser le nombre de sous-graphes visités en dehors des sous-graphes énumérés fréquents fermés. Pour illustrer l'impact de cet investissement de la forte accessibilité, une comparaison par rapport à un algorithme qui parcoure tout l'espace de recherche avant de récupérer les fermés sera illustrée.

Enfin, nous présentons les résultats expérimentaux de deux algorithmes proposés pour la découverte des sous-graphes connexes à étiquetages d'arêtes non redondantes fréquents fermés : **ECERCSgl** (**E***dges and* **C***onverses* **E***dges* **R***epresentation* **C***losed* **S***ub−* **g***raphs* **l***ist*), et **ADARCSgl** (**AD***jacency* **A***rray* **R***epresentation* **C***losed* **S***ub−***g***raphs* **l***ist*).

Ces deux algorithmes visent la réduction des sous-graphes parcourus et de minimiser le coût des tests d'isomorphisme de graphes et des sous-graphes.

4.1 Représentation des graphes

Comme nous l'avons mentionné au niveau de la sous-section 3.3, il existe plusieurs représentations des graphes : Matrices d'adjacences, listes d'adjacences, tableaux d'adjacences, etc. Ces représentations influencent directement le coût d'énumération des sous-graphes connexes. Théoriquement, la complexité du délai d'énumération d'un sous-graphe connexe est indiquée à base de la complexité du nombre de tests d'isomorphismes de sous-graphes effectués pour calculer la fréquence. Cependant, en pratique la complexité est un cumul des temps investis pour l'énumération de chacun des sous-graphes. Autrement dit, le délai d'énumération est la somme des temps consommés lors des différentes opérations, soient élémentaires (test d'isomorphisme de graphes et de sous-graphes), ou secondaires (augmentation (jonction) et test de connexité).

La représentation des graphes est une étape clé pour le développement d'un algorithme de découverte des sous-graphes. Au niveau de cette sous-section, nous présentons deux représentations des graphes par rapport à la classe des graphes à étiquetage d'arêtes non redondant et nous allons illustrer leurs impacts vis-à-vis les différentes opérations.

4.1.1 Représentation ensembliste

Généralement, les algorithmes de découverte des sous-graphes connexes fréquents considèrent des bases de graphes (c.-à-d. un ensemble de graphes) [Nijssen and Kok2005] [Yan and Han2002] [Inokuchi *et al.*2000] [Kuramochi and Karypis2001] [Yan and Han2003]. Au niveau des bases de graphes une transaction (un graphe) est représentée par l'indice de la transaction (p. ex. t0), chacun des sommets est trié dans l'ordre de son indice (p. ex. v 0 0, v 1 0, v 2 5,..., v n 0), et chacune des arêtes (p. ex. e 0 1 6), ce qui indique que l'arête reliant les sommets ayant les indices 0 et 1, et l'étiquette de cette arête est de 6.

La représentation ensembliste des graphes n'a pas été une préférence pour le développement d'algorithmes de découvertes. Après l'apparition des bibliothèques de gestion des ensembles tels que : STL [Plauger *et al.*2000] pour C++ et Java Collections-Set [Collins2011] pour java, l'utilisation de la représentation ensembliste est de plus en plus répondue [Thomas *et al.*2010], [Thomas *et al.*2009]. Pour le cas de la classe des graphes à étiquetage d'arêtes non redondant nous avons opté pour cette représentation motivée par :

- La facilité de codage des graphes de la base de données par une représentation ensembliste.
- La non-redondance des étiquettes des arêtes et les indices des sommets permettent de générer des ensembles de sommets et d'arêtes totalement ordonnés, ce qui facilite les opérations d'augmentation et de test de connexité, ainsi que le test d'équivalence et celui d'isomorphisme de sous-graphes.
- La facilité d'encoder et de décoder les graphes comme et à partir des ensembles d'items d'arêtes et d'arêtes converses (c.-à-d. ECE).

La représentation ensembliste consiste à ce que les graphes sont représentés en mémoire comme des ensembles théoriques c.-à-d. un couple d'ensembles : l'un des sommets et l'autre des arêtes $G(V, E)$ où chacun des sommets est représenté comme un couple (i, et_v) et chacune des arêtes par un triplet (i, et_e, j). Pour les sommets, i indique l'indice du sommet, généralement un entier, et et_v indique l'étiquette du sommet. Pour les arêtes, les indices i et j indiquent les deux sommets aux indices i et j, et et_e indique l'étiquette de l'arête. La figure 4.1 illustre un exemple de deux graphes connexes à étiquetage d'arêtes non redondant à gauche, et leurs représentations ensemblistes correspondantes à droite.

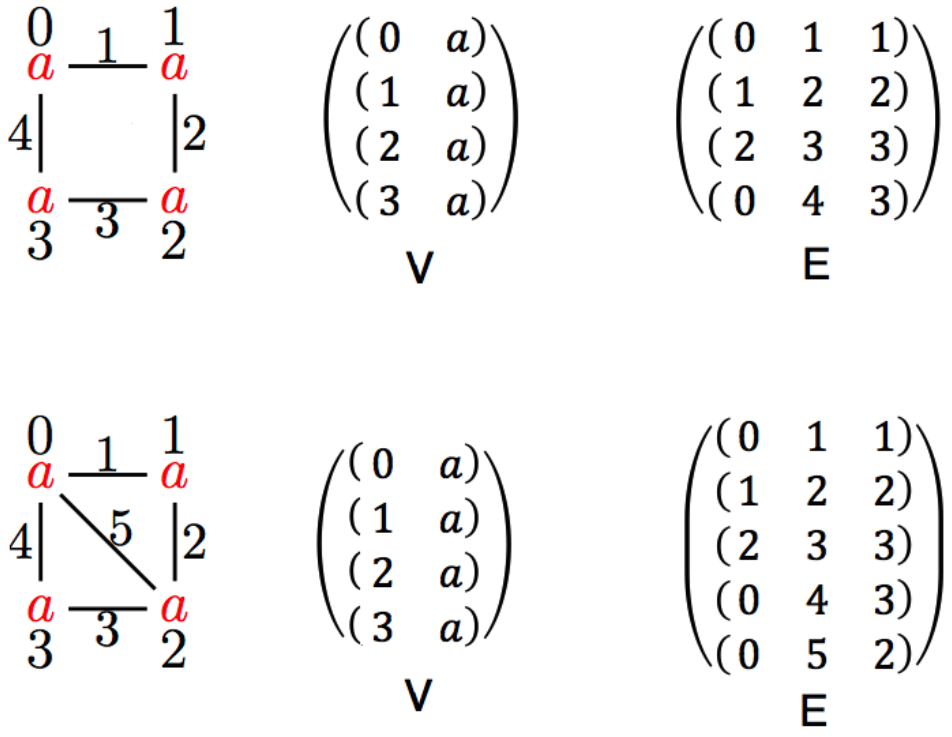

FIGURE 4.1 – Exemple de la représentation ensembliste.

Cette représentation ensembliste permet de tester la connexité d'une nouvelle arête e à un graphe connexe g en cherchant l'existence de l'un de ces indices i, j au niveau de l'ensemble des sommets V de g. Cette opération coûte au pire des cas $\mathcal{O}(|V(g)|^2)$, sachant que les ensembles des sommets sont ordonnés par rapport à un ordre croissant des indices la recherche peut être achevée dans $\mathcal{O}(|V(g)|)$ au pire des cas. Pour l'opération d'augmentation il suffit d'ajouter l'arête e à l'ensemble des arêtes $g(E)$ et les deux sommets correspondants à l'ensemble des sommets $g(V)$. Cette dernière opération consiste à chercher pour une arête e les deux sommets correspondants au niveau d'un graphe de la base de graphes $\mathcal{D}$, ensuite, d'ajouter soit les deux sommets pour le cas de $g = \emptyset$, l'un des deux si l'autre est déjà existant, aucun des deux s'ils sont déjà bien existants.

Pour le test d'équivalence (c.-à-d. l'isomorphisme de graphes), ou l'isomorphisme de sous-graphes entre deux graphes le coût au pire des cas est $\mathcal{O}(2^{|E|})$, où pour chacune des arêtes, nous pouvons avoir deux plongements par rapport à ses sommets. Cependant, l'ordre des ensembles des sommets et des arêtes permettent d'optimiser ce coût en pratique.

4.1.2 Représentation par tableaux d'adjacences

Une représentation par tableaux d'adjacences consiste à représenter un graphe par deux tableaux. Ces deux tableaux contiennent les sommets et les arêtes qui les relient comme suit : les sommets du graphe sont sauvegardés par couples au niveau du deuxième tableau, où chaque couple consiste à deux sommets x_1, et x_2 reliés par une arête y. Au niveau du premier tableau, les positions des premiers sommets de chaque couple du tableau 2 sont sauvegardées aux positions des étiquettes d'arêtes correspondantes. Autre que ce cas de non-redondance des étiquettes des arêtes. Cette représentation ne peut pas être utilisée. Le choix de cette représentation est motivé principalement par la facilité d'accessibilité vers le couple des sommets de chacune des arêtes. La figure 4.2 illustre un graphe connexe à étiquetage d'arêtes non redondant (a), sa représentation comme tableaux d'adjacences (b), le graphe après augmentation (c), et la représentation correspondante au graphe ayant été augmenté (d).

Un tel principe de représentation peut être vu comme un map où les clés sont les étiquettes des arêtes et la valeur de chaque clé et un tableau contenant les deux sommets de l'arête ayant la clé en question comme étiquette. La considération d'un map au lieu de tableaux d'adjacences permet de conserver l'aspect fonctionnel, et aide même à l'améliorer. La figure 4.3 illustre la représentation du graphe connexe à étiquetage d'arêtes non redondant sur la figure 4.2 (a) comme un map.

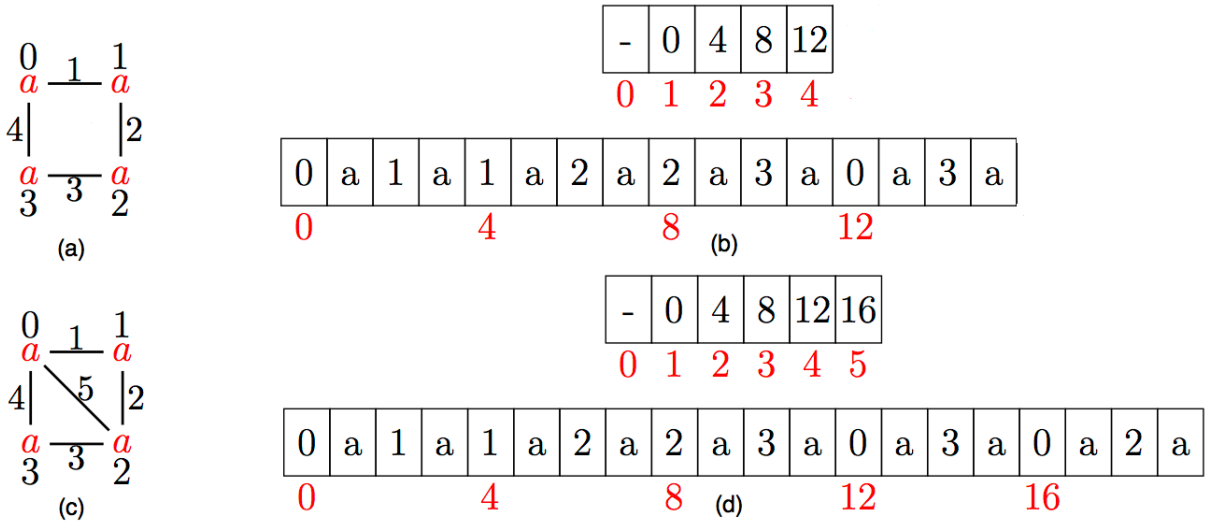

FIGURE 4.2 – Exemple d'augmentation de tableaux d'adjacences.

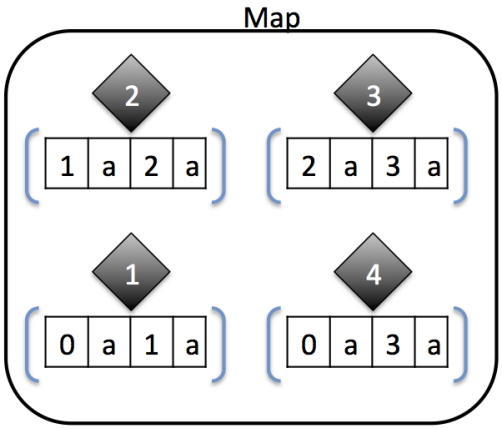

FIGURE 4.3 – Exemple de représentation en map du graphe (a) de la figure 4.2.

Cette représentation en map facilite les tests de connexité et d'augmentation. Pour tester si une nouvelle arête e est connexe à un graphe connexe g par rapport à un graphe $G \supset g$ il suffit de chercher une clé (arête) e_2 qui partage un sommet avec e. Pour l'augmentation, l'opération est beaucoup plus simple à partir du fait que e est connexe à g, il suffit d'ajouter le tableau des sommets de e avec l'étiquette de e comme clé au map.

Encore, ce dernier type de représentation permet de tester d'isomorphisme de graphes et de sous-graphes plus rapidement pour plusieurs cas. La résolution des tests d'isomorphisme de graphes et de sous-graphes est d'un ordre $\mathcal{O}(|E| * 2^{\mu})$, où μ n'est que le nombre des arêtes ayant les deux étiquettes de ses deux sommets identiques. Donc, pour le meilleur des cas le problème peut être résolu dans un temps linéaire.

4.1.3 Expérimentations

Pour avoir une idée sur les temps d'exécutions des opérations sur les graphes à étiquetage d'arêtes non redondant. Nous présentons une comparaison des résultats des temps d'achèvement des tests d'isomorphisme de graphes par rapport à des configurations diverses des graphes pour les deux représentations des graphes : ensembliste et tableaux d'adjacences.

Pour mener cette expérience, nous avons généré des graphes connexes de différentes tailles, d'ordres et de densités. Pour la génération des graphes, nous avons développé un générateur de graphes connexes à étiquetage d'arêtes non redondant noté GenUniqueEdgesLabelsGraph. Pour la phase de génération, nous avons considéré 5 paramètres, S la taille des graphes (c.-à-d. le nombre d'arêtes), ce paramètre a été défini comme une fonction aléatoire ayant comme image un entier au sein de l'intervalle [S-5, S+5]. ETV,ETE le nombre d'étiquettes des sommets et des arêtes qui peuvent être considérés, respectivement. L représente la densité des graphes, comme l'une des trois valeurs suivantes : 0, 0.5, et 1, le graphe est complet si $L = 1$, alors il est totalement sparse pour $L = 0$, et pour $L = 0.5$, il est d'une densité moyenne. La génération d'un graphe d'une taille S donnée est procédé comme suit : d'abord le nombre de sommets $|V|$ est déterminé selon la taille des graphes et la densité souhaitée par exemple pour une densité $L = 0$, $|V| = |E| + 2$ ou $|V| = |E| + 1$. Ce calcul du nombre de sommets pour ce dernier cas est décidé par rapport à ce que les graphes les moins denses ne sont que des trous ou des chaines de tailles $|E|$. Pour une densité $L = 1$, le nombre de sommets n'est que le plus grand entier x qui permet de vérifier l'équation suivante : $|E| = \frac{x*(x-1)}{2}$. Alors que pour une $L = 0.5$, le nombre de sommets pour le cas de densité $L = 1$ (c.-à-d. x) additionné à un nombre aléatoire de 1 à x. Après avoir fixé le nombre de sommets, $|V|$ sommets sont générés et S liens entre ces derniers sont ajoutés afin de couvrir le maximum de ces sommets. Enfin chacun des graphes générés est nettoyé des sommets qui n'ont pas été reliés à aucun autre sommet.

Les bases de graphes générés pour mener cette expérience sont composées de 10 graphes pour chacune des configurations souhaitées. Ensuite, pour chacune des bases de graphes nous générons une base des graphes duplicata $\mathcal{D}'$ en appliquant des permutations de

sommets et des indices des sommets afin d'avoir les cas les plus proches possible du pire des cas de test d'isomorphisme de graphes. Les paramètres appliqués lors de la génération des bases de graphes sont : 1) S prend les valeurs 10, et 20 ce qui permet d'avoir des tailles de graphes entre $[5, 25]$. 2) Le nombre des étiquettes des sommets ETV et des arêtes ETE est varié : a) entre 1, et 10 pour les sommets V, et b) entre 16, 26, et 50 pour les arêtes E. 3) La densité des graphes générés est de 0 ,0.5 et 1. Ces variations des paramètres ont comme but de générer des cas de base de graphes différents, afin d'avoir une meilleure image des comportements des tests effectués. La table 4.1 illustre les résultats obtenus à partir des différentes bases de graphes qui ont été générées par rapport à une taille de $S = 10$ les temps présentés représentent d'une part, le temps total des temps en secondes d'exécutions des tests d'isomorphismes de graphes par rapport à chacun des graphes $g \in \mathcal{D}$ à l'égard de chacun des graphes de la base des graphes duplicatas $\mathcal{D}'$. Ce dernier temps est noté le temps d'isomorphisme de graphe total. D'autre part, le temps total des temps d'exécutions des tests d'isomorphismes de graphes par rapport à chacun des graphes et son duplicata au sein de la base des graphes duplicatas $\mathcal{D}'$ (c.-à-d. $\forall g \in \mathcal{D}, g' \in \mathcal{D}'$ la somme des temps des tests d'isomorphismes de graphes entre g et g' t.q g' est un graphe duplicata de g). Ce dernier temps est noté le temps d'isomorphisme de graphes un-à-un. Nous allons aussi noter la représentation ensembliste par ENS, celle par tableaux d'adjacences par ADA.

TABLE 4.1 – Les résultats des tests d'isomorphisme de graphes de tailles 10.

Bases de graphes	*Temps isg total*		*Temps isg un* à *un*	
Id	ENS	ADA	ENS	ADA
T10S10ETV1ETE16L0.0	0.386	0.207	0.225	0.203
T10S10ETV1ETE16L0.5	0.335	0.206	0.214	0.203
T10S10ETV1ETE16L1.0	0.294	0.189	0.218	0.187
T10S10ETV1ETE50L0.0	0.363	0.215	0.237	0.212
T10S10ETV1ETE50L0.5	0.365	0.21	0.229	0.21
T10S10ETV1ETE50L1.0	0.314	0.197	0.246	0.196
T10S10ETV10ETE16L0.0	0.287	0.177	0.211	0.175
T10S10ETV10ETE16L0.5	0.287	0.161	0.18	0.161
T10S10ETV10ETE16L1.0	0.248	0.157	0.19	0.156
T10S10ETV10ETE50L0.0	0.288	0.179	0.199	0.17
T10S10ETV10ETE50L0.5	0.301	0.169	0.192	0.168
T10S10ETV10ETE50L1.0	0.215	0.167	0.181	0.167

En suivant les résultats de la table 4.1, nous remarquons facilement la domination générale de la représentation par tableaux d'adjacences où les résultats sont totalement

en faveur de ces derniers par rapport à la représentation ensembliste. Il est encore clair de signaler que pour le cas de la représentation ensembliste, pratiquement 1/3 des temps d'exécutions est dédié aux cas de décider que les graphes ne sont pas isomorphes, contrairement au cas de représentation ADA au le temps de décider que les graphes ne sont pas isomorphes est pratiquement négligeable. Nous remarquons aussi que l'augmentation de densité réduit les temps des tests d'isomorphismes de graphes, cela est dû à la réduction du nombre de sommets. La table 4.2 illustre les résultats obtenus à partir des différentes bases de graphes qui ont été générés par rapport à une taille de $S = 20$.

TABLE 4.2 – Les résultats des tests d'isomorphisme de graphes de tailles 20

Bases de graphes	*Temps isg total*		*Temps isg un* à *un*	
Id	*ENS*	*ADA*	*ENS*	*ADA*
T10S20ETV1ETE26L0.0	367.1	47.101	106.266	46.881
T10S20ETV1ETE26L0.5	147.013	23.668	40.674	20.913
T10S20ETV1ETE26L1.0	19.626	3.281	4.726	3.25
T10S20ETV1ETE50L0.0	227.326	30.14	75.425	29.749
T10S20ETV1ETE50L0.5	395.685	68.01	137.377	66.684
T10S20ETV1ETE50L1.0	6.63	2.37	2.83	2.336
T10S20ETV10ETE26L0.0	228.253	34.332	70.418	34.009
T10S20ETV10ETE26L0.5	30.5	7.736	11.054	6.969
T10S20ETV10ETE26L1.0	23.492	5.211	8.084	4.919
T10S20ETV10ETE50L0.0	252.575	38.035	84.974	38.015
T10S20ETV10ETE50L0.5	41.532	8.974	16.533	8.728
T10S20ETV10ETE50L1.0	14.103	3.101	4.806	3.093

Semblable aux résultats de la table 4.1, les résultats illustrés au niveau de la table 4.2 confirment la supériorité de la représentation ADA. L'augmentation de la taille des graphes à 20 a permis de mettre l'emphase sur la différence des temps de calcul entre la représentation ensembliste et celle ADA. Ces résultats confirment nos études de complexités discutées auparavant qui ont favorisé la représentation ADA par rapport à l'ensembliste. Encore pour cette table il est facile de vérifier que pour le cas de la représentation ensembliste que pratiquement 2/3 des temps d'exécutions sont dédiés aux cas de décider que les graphes ne sont pas isomorphes, ce qui prouve la souffrance de test vis-à-vis la recherche d'arêtes et de sommets au niveau des ensembles d'arêtes et de sommets respectivement. Nous remarquons encore aussi que l'augmentation de densité réduit les temps des tests d'isomorphismes de graphes.

4.2 Codage

Généralement, être placé devant un nouveau problème, la première intention est de le ramener à un autre déjà rencontré. La même attitude est adoptée pour pouvoir plus facilement manipuler les ordres, il est parfois intéressant d'utiliser des fonctions dont on maîtrise mieux l'image tout en garantissant la conservation de l'ordre.

Definition 4.2.1 *(Codage) Soient $P = (X, \leq_P)$ et $Q = (Y, \leq_Q)$ deux ensembles ordonnés et f une application de P dans Q. L'application f est dite un codage (plongement) de P dans Q si elle vérifie : pour tous x, $x^{'} \in P$,*

$$x \leq_P x^{'} \iff f(x) \leq_Q f(x^{'}).$$

Dans ce dernier contexte, nous investissions un encodage canonique des graphes à étiquetage d'arêtes non redondant qui a été proposé par [Thomas *et al.*2009] (nous notons par encodage (code) l'ensemble d'items qui représente un graphe donné, et un codage, la représentation de tout l'espace de recherche). Nous avons entamé cet encodage au niveau de la section 3.3.2. Alors que cet encodage permet de résoudre le problème d'isomorphisme de graphes, il ne permet pas de préserver les non-comparabilités, où deux encodages de deux graphes connexes non comparables pouvant être comparables. En utilisant cet encodage comme fonction de codage $ECE : \mathcal{L}(G) \rightarrow \mathcal{Z}(G)$, où $\mathcal{L}(G)$ est l'ensemble des sous-graphes connexes d'un graphe G et $\mathcal{Z}(G)$ l'ensemble des parties de l'encodage ECE du graphe G, certaines relations de non-comparabilités ne sont plus préservées. Cette perte affectera les résultats de fréquence et l'ordre des codes candidats. Donc, le codage proposé est défini comme suit : pour tous g, g$^{'}$ $\in \mathcal{L}(G)$,g $\subseteq g' \Rightarrow ECE(g) \subseteq_Q ECE(g')$.

4.2.1 La faille de codage

Comme un encodage d'un graphe g de taille 3 et plus, en un ensemble d'items d'arêtes et d'arêtes converses, est l'ensemble des encodages de ces sous-graphes connexes $\mathcal{E}$ de tailles 3. Cependant, pour deux graphes non comparables l'ensemble des sous-graphes $\mathcal{E}$ de l'un des deux peut être inclus au niveau de l'autre. Ce phénomène (faille de codage) est lié aux graphes ayant plus que deux sous-graphes de tailles 3 de types chaines linéaires. Comme nous pouvons le constater sur la figure 4.4 le graphe g_1 sur la partie (a) et celui sur la partie (b) ne sont pas isomorphes alors que les deux sous-graphes $\mathcal{E}(g_1)$ ((a).a et (a).b) sont isomorphes par rapport à deux des sous-graphes $\mathcal{E}(g_2)$ ((a).a $\cong$ (b).a et (a).b $\cong$ (b).b). Alors en encodant le graphe g_1 et g_2 par ECE l'ensemble d'items du premier sera inclus au niveau du code du deuxième, alors que g_1 n'est pas sous-graphe isomorphe de g_2.

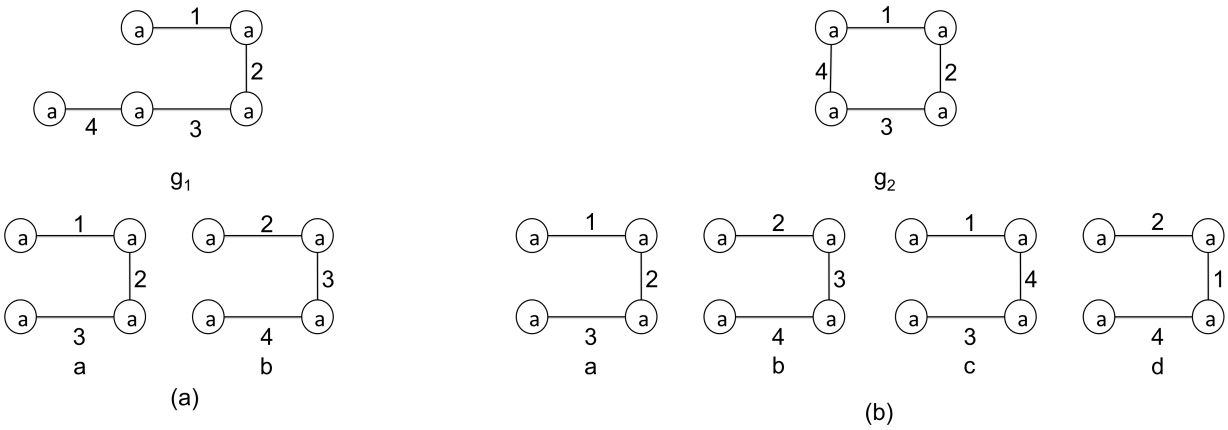

FIGURE 4.4 – Exemple d'une faille de codage.

Le fait de traiter chaque graphe G_i de la base des graphes $\mathcal{D}$ séparément et que l'étiquetage des arêtes est non redondant permettent de contourner les fausses comparabilités au sein d'un codage $\mathcal{Z}(G_i)$ d'un espace de recherche $\mathcal{L}_i$, cela est dû à l'étiquetage non redondant des arêtes qui ne permet pas d'avoir deux sous-graphes aptes à produire une faille de codage. Cependant, cette faille de codage affectera le calcul de la fréquence si l'inclusion d'ensembles est adoptée comme opérateur de comparabilité (c.-à-d. relation d'ordre). À titre d'exemple, le graphe g_1 sur la partie (a) de la figure 4.4 sera inclus au niveau de tous les graphes $G_i \in D$ incluent le graphe g_2 sur la partie (b) alors que ce n'est pas le cas par rapport à l'isomorphisme de sous-graphes. Donc, pour une base de graphes $\mathcal{D}$ qui contient deux graphes G_1 et G_2 à étiquetage d'arêtes non redondant, qui incluent g_1 et g_2 comme sous-graphes respectivement. Pour un seuil de fréquence de 2, nous obtiendrons le graphe g_1 comme fréquent alors qu'il ne se présente qu'une seule fois au niveau du premier graphe G_1.

4.2.2 Détection de la faille de codage

Nous pouvons facilement remarquer que la faille de codage dépend fortement des sous-graphes de tailles 3 (c.-à-d. $\mathcal{E}$). Une faille de codage par rapport à deux graphes non comparables g et $g^{'}$ est une conséquence logique d'avoir tous les sous-graphes $\mathcal{E}(g)$ inclus au niveau des sous-graphes $\mathcal{E}(g^{'})$. Plus précisément au sein de deux graphes g et g' respectivement deux sous-graphes s et s' ayant la même taille, composés des mêmes arêtes et $\mathcal{E}(s) \subseteq \mathcal{E}(s')$, mais $s \nsubseteq s'$ (s et s' sont non comparable). Ce dernier phénomène ne peut être atteint que pour des cas où le premier sous-graphe (c.-à-d. s) est une chaine (succession de chaines linéaires) qui commence et se termine à deux sommets distincts ayant des étiquettes identiques et le second (c.-à-d. s') est un cycle composé des mêmes

arêtes de la chaine s dans le même ordre, mais commence et se termine à un même sommet.

La refermeture de la dernière arête d'une chaine vers le sommet du commencement génère deux nouveaux sous-graphes de taille 3 (c.-à-d. $\in \mathcal{E}$) de types chaines linéaires. Par conséquent, pour détecter la faille de codage au niveau d'un graphe g par rapport un autre g' : il faut d'abord chercher les couples d'arêtes suspectes qui sont identifiables par les caractéristiques suivantes : a) deux arêtes non reliées l'une à l'autre impliquent automatiquement qu'au niveau du code il n'existe pas d'arête converse qui représente la relation entre les deux. b) Deux arêtes ne peuvent pas se retrouver au sein d'un même triplet connexe (c.-à-d. un code de trois arêtes connexe). Ensuite, tester si les deux arêtes de l'un des couples d'arêtes suspectes sont bien reliées au niveau du code du deuxième graphe g'. Si aucune des arêtes n'est reliée alors il n'existe pas de faille de codage sinon il s'agit bien d'une faille.

4.2.3 Expérimentations

Pour avoir une idée des temps d'exécutions en secondes des opérations d'encodage et des tests d'inclusions des graphes à étiquetage d'arêtes non redondant, nous présentons une comparaison des résultats des temps d'exécutions par rapport à une représentation ECE des graphes. Les tables 4.3, 4.4, et 4.5 illustrent les temps consommés pour tester l'équivalence des codes des graphes : a) en considérant le test de faille de codage, noté atf et b) sans la considération du test de faille de codage, noté stf. Ces résultats sont issus à une expérimentation par rapport à un ensemble de base de graphes ayant été générés à base de configurations de taille : 20 et 30 et un nombre d'étiquettes de sommets 1, 10, et 50. Le nombre d'étiquettes d'arêtes est de 26, 36, 50, et 70. La densité est de : 0, 0.5, et 1. Pour cette expérience les temps d'exécutions sont des temps en secondes de type un-à-un similaire à l'expérience au niveau de la sous section 4.1.3.

Les résultats présentés au niveau de la table 4.3 illustrent une légère supériorité des temps issus par rapport aux tests sans la considération de la faille de codage (c.-à-d. atf). Ces derniers résultats ont été attendus vis-à-vis du fait que le test d'existence de faille de codage nécessite un temps supplémentaire, mais par rapport aux configurations des bases de taille 20 testées, nous pouvons facilement remarquer que le temps de test de faille de codage n'a pas une grande influence sur le temps total du test (c.-à-d. atf). Autre que la comparaison des temps stf et atf, la table 4.3 illustre les temps de calculs de l'isomorphisme de graphes par rapport à une représentation ADA, et nous remarquons clairement que ces derniers tests consomment largement beaucoup plus de temps (plus que 100 fois) que les tests d'équivalence d'ensemble de code ECE de graphes soit : en

TABLE 4.3 – Les résultats des tests d'isomorphisme de graphes de ECE et ADA par rapport à une taille de 20.

Bases de graphes	*Temps de test d'isomorphisme*		
Id	stf	atf	ADA
T10S20ETV1ETe26L0.0	0.136	0.211	46.881
T10S20ETV1ETe26L0.5	0.165	0.169	20.913
T10S20ETV1ETe26L1.0	0.188	0.196	3.25
T10S20ETV1ETe50L0.0	0.113	0.155	29.749
T10S20ETV1ETe50L0.5	0.143	0.215	66.684
T10S20ETV1ETe50L1.0	0.157	0.165	2.336
T10S20ETV10ETe26L0.0	0.143	0.154	34.009
T10S20ETV10ETe26L0.5	0.15	0.167	6.969
T10S20ETV10ETe26L1.0	0.174	0.181	4.919
T10S20ETV10ETe50L0.0	0.143	0.147	38.015
T10S20ETV10ETe50L0.5	0.149	0.166	8.728
T10S20ETV10ETe50L1.0	0.167	0.17	3.093

considérant le test de faille de codage (c.-à-d. atf), ou non (c.-à-d. stf). Au niveau de la table 4.4 les bases de données ont été configurées pour des tailles de graphes de 30, avec un niveau moyen de permutation, et de changement d'identifiants de sommets des duplicatas.

TABLE 4.4 – Les résultats des tests d'isomorphisme de graphes de ECE par rapport à une taille de 30.

Bases de graphes	*Temps de test d'isomorphisme*		
Id	stf	atf	ADA
T10S30ETV1ETe36L0.0	0.238	0.269	2.581
T10S30ETV1ETe36L0.5	0.335	0.336	1.693
T10S30ETV1ETe36L1.0	0.404	0.411	0.795
T10S30ETV1ETe70L0.0	0.257	0.266	3.145
T10S30ETV1ETe70L0.5	0.339	0.357	1.952
T10S30ETV1ETe70L1.0	0.389	0.402	0.761
T10S30ETV10ETe36L0.0	0.215	0.213	2.31
T10S30ETV10ETe36L0.5	0.319	0.331	1.973
T10S30ETV10ETe36L1.0	0.316	0.336	0.735
T10S30ETV10ETe70L0.0	0.225	0.235	3.1
T10S30ETV10ETe70L0.5	0.35	0.352	2.014
T10S30ETV10ETe70L1.0	0.342	0.33	0.841

Les résultats présentés au niveau de la table 4.4 confirment les résultats de la table 4.3, mais nous remarquons une large amélioration des temps des tests ADA malgré l'augmentation de la taille des graphes. Cependant, les temps issus des tests d'équivalences des encodages sont largement meilleurs. Au niveau de la table 4.5 les bases de données ont été configurées pour des tailles de graphes de 50, avec juste des changements d'identifiants de sommets des duplicatas.

TABLE 4.5 – Les résultats des tests d'isomorphisme de graphes de ECE par rapport à une taille de 50.

Bases de graphes	*Temps de test d'isomorphisme*		
Id	stf	atf	ADA
T10S50ETV5ETe56L0.0	0.232	0.47	0.078
T10S50ETV5ETe56L0.5	0.346	1.0	0.065
T10S50ETV5ETe56L1.0	0.402	1.272	0.056
T10S50ETV5ETe100L0.0	0.239	0.459	0.074
T10S50ETV5ETe100L0.5	0.364	1.194	0.077
T10S50ETV5ETe100L1.0	0.401	1.258	0.053
T10S50ETV10ETe56L0.0	0.212	0.422	0.074
T10S50ETV10ETe56L0.5	0.352	0.958	0.069
T10S50ETV10ETe56L1.0	0.413	0.984	0.053
T10S50ETV10ETe100L0.0	0.251	0.387	0.076
T10S50ETV10ETe100L0.5	0.305	1.019	0.07
T10S50ETV10ETe100L1.0	0.393	1.066	0.056

La table 4.5 illustre une supériorité de performance de la représentation ADA des graphes vis-à-vis des encodages ECE. Ces résultats ont été attendus par rapport à la complexité linéaire du test d'isomorphisme de graphes pour le meilleur des cas en respectant une représentation ADA des graphes. Nous distinguons aussi au niveau de cette dernière table que le temps des tests atf est largement plus coûteux que le temps stf. De même pour l'augmentation du niveau de densité des graphes, les temps des tests augmentent systématiquement. Ce phénomène indique clairement que les complexités de la détection de la faille de codage, le test d'équivalence de codes d'items, et l'encodage augmentent par rapport à l'augmentation des tailles et des densités des graphes.

4.3 L'espace de recherche

Autre que les principales opérations sur les graphes qui consistent à tester l'isomorphisme de graphes et de sous-graphes afin de vérifier la redondance, ainsi que de calculer

la fréquence, la réduction du nombre de candidats à visiter au sein de l'espace de recherche permet de booster considérablement l'efficacité d'un processus de découverte de sous-graphes connexes fréquents fermés, ou maximaux. La principale motivation d'énumération des motifs fréquents fermés et maximaux est leurs nombres, qui sont largement réduits par rapport à la cardinalité de l'ensemble de tous les motifs fréquents, ainsi que ce dernier ensemble de tous les fréquents peut être reconstruit à partir de l'ensemble des fermés et ceux des maximaux par rapport à la fermeture descendante de la fréquence.

Dans cette section, nous étudions le système des sous-graphes connexes d'un graphe à étiquetage d'arêtes non redondant. Pour ce travail nous allons considérer pour chaque graphe $G_i \in \mathcal{D}$ un espace de recherche $\mathcal{L}_i$ qui est composé des sous-graphes connexes rangés suivant leurs tailles (nombre d'arêtes), où chaque sous-graphe est rangé au rang de sa taille, le vide ($\emptyset$) étant au rang 0. Deux sous-graphes $x, y \in \mathcal{L}_i$ sont dits comparables $x \subseteq_{sgi} y$ si et seulement si l'ensemble d'arêtes $E(x)$ de x est contenu dans celui $E(y)$ de y c'est-à-dire $x \subseteq y \Leftrightarrow E(x) \subseteq E(y)$. Cette comparabilité est due à la contrainte d'étiquetage d'arêtes non redondant ce qui nous permettra d'investir la notion d'identité. Par conséquent, le langage $\mathcal{L}_i$ de chacun des graphes $G_i(V_i, E_i)$ de $\mathcal{D}$ est défini comme suit :

$$\mathcal{L}_i = \{X(V_X, E_X) \in G_i \mid E(X) \textit{ est connexe}\}$$

À partir de cette définition, nous pouvons illustrer l'espace de recherche $\mathcal{L}(G)$ d'un graphe $G(V, E)$ comme étant tous les sous-graphes g ayant des ensembles d'arêtes $E(g) \subseteq E(G)$ connexes sur G. Autrement dit, tous les sous-graphes ayant un ensemble d'arêtes $E_i \subseteq E(G)$ sauf ceux formant des sous-graphes non connexes. Donc, l'espace de recherche peut être défini comme suit :

Definition 4.3.1 *($\mathcal{L}_i$ l'ensemble des sous-graphes connexes d'un graphe G_i) Pour un graphe connexe $G_i = (V_i, E_i)$ et X un sous-ensemble d'arêtes tels que $X \subseteq E_i$, $G_i[X]$ est le sous-graphe induit par X au niveau de G_i. Le système des sous-graphes connexes $\mathcal{L}_i$ de G_i est composé de tous les $X \in \mathcal{P}ow(E_i)$ tels que $\mathcal{P}ow(E_i)$ est l'ensemble de tous les sous-ensembles de E_i et X est un ensemble d'arêtes connexes X au niveau de G_i.*

$$\mathcal{L}_i = \{X \in E_i \mid G_i[X] \textit{ est connexe}\}$$

Comme nous l'avons noté auparavant, les propriétés structurelles de l'espace de recherche jouent un rôle clé pour l'adoption des manoeuvres d'optimisations, afin d'atteindre l'objectif de trouver les motifs fréquents fermés ou maximaux au moindre coût. Cette réduction des coûts revient à la réduction des coûts des opérations de générations et de

tests de fréquence des candidats, ainsi que le nombre de ces candidats. Alors que la réduction des coûts des opérations est liée à la complexité des motifs manipulés, la réduction du nombre des candidats permet d'optimiser le nombre d'appels à ces opérations. Cette optimisation est liée aux propriétés associées à la relation d'ordre au sein de l'espace de recherche et les motifs composent ce dernier.

Généralement, les algorithmes qui optimisent au maximum le nombre des candidats pour l'énumération des motifs fréquents fermés et maximaux exploitent des espaces de recherches ayant des formes structurelles particulières. Pour les motifs fréquents maximaux, les algorithmes les plus efficaces investissent le principe de dualisation qui nécessite que l'espace de recherche contienne le complément de chacun des motifs, ce qui nécessite que cet espace soit isomorphe à un treillis. À titre d'exemple, nous citons les itemsets, les clés [Gunopulos *et al.*2003], les séquences rigides [Nourine and Petit2012], etc. Pour le cas des motifs fréquents fermés, la forte accessibilité de l'espace de recherche est indispensable pour réduire au maximum les candidats générés avant d'atteindre tous les fermés [Boley *et al.*2010].

Pour atteindre notre objectif de trouver les fermés en minimisant le nombre des sous-graphes visités, notre système des sous-graphes $\mathcal{L}_i$ pour le cas des graphes à étiquetage d'arêtes non redondant doit être fortement accessible. Cette forte accessibilité permettra de sauter d'un sous-graphe fermé c qui a été déjà énuméré vers ces successeurs fermés par l'augmentation du premier (c.-à-d. générer un nouveau sous-graphe connexe $g' = c \cup e$ tels que $e \in E(G_i) \backslash E(c)$) et en calculant l'opérateur de fermeture sur ce dernier $\sigma(g')$ s'il existe.

Theorem 4.3.1 *Le système $\mathcal{L}_i$ est fortement accessible.*

Proof 4.3.1 *$\mathcal{L}_i$ est accessible, soit un graphe $g \in \mathcal{L}_i$ si g est un graphe qui admet un cycle t alors le fait de retirer une arête e de ce cycle t, un autre graphe connexe est généré $g' = g \backslash e \in \mathcal{L}_i$ sinon si g est sans cycle, il suffit de retirer une arête e avec son sommet feuille (c.-à-d. supprimer ($e(u, v)$ et v) avec v un sommet d'extrémité de degré un), pour avoir encore un graphe $g' = g \backslash e \in \mathcal{L}_i$.*
Soit deux graphes $X_1, X_2 \in \mathcal{L}_i$ avec $X_2 \subset X_1$. Nous supposons qu'il n'existe pas une arête $e \in X_1 \backslash X_2$ tel que $X_2 \cup e \in \mathcal{L}_i$. Donc X_2 et $X_1 \backslash X_2$ ne sont pas connectés dans G_i contradiction avec le choix de X_2 comme un graphe connexe.

Autrement dit, un système est fortement accessible, si chaque $X \in \mathcal{L}_i$ peut être atteint à partir de tous $Y \subset X$ avec $Y \in \mathcal{L}_i$ via des augmentations à un élément. Ce principe va

être la clé de la réduction du nombre des sous-graphes parcourus, où pour chaque graphe fermé c_2 s'il existe un autre fermé c_1 et $c_2 \subset c_1$ alors c_1 sera atteint à partir de c_2 en ajoutant une arête e et en calculant la fermeture $c_1 = \sigma(c_2 \cup e)$.

4.4 Le calcul de la fermeture

Autre que l'accessibilité forte du système qui garantie l'existence d'une chaine entre chaque couple de sous-graphes connexes fermés successifs, le calcul de fermeture permet de sauter directement à partir d'un sous-graphe successeur immédiat d'un sous-graphe fermé c_1 vers l'un de ces sous-graphes fermés successeurs c_2. Pour le cas des itemsets, la fermeture d'un ensemble d'items i consiste en l'intersection des transactions qui contient i comme sous-ensemble [Uno *et al.*2004] [Boley *et al.*2010]. Malheureusement, pour le cas du système des sous-graphes connexes $\mathcal{L}_i$ le résultat obtenu à l'issue du calcul de la fermeture d'un sous-graphe connexe donné n'est pas unique et peut être un graphe déconnecté qui n'appartient même pas au système. Par conséquent, nous déduisons qu'il n'existe pas un opérateur de fermeture associe au système $\mathcal{L}_i$. Cependant, nous définissons une opération de fermeture de fréquence au sein de $\mathcal{L}_i$ par rapport $\mathcal{D}$ comme suit :

Definition 4.4.1 *(opération de fermeture) Soit un système $\mathcal{L}_i$ des sous-graphes connexes d'un graphe G_i, et une base de graphes connexes $\mathcal{D}$. L'opération de fermeture au sein de $\mathcal{L}_i$ en accord avec $\mathcal{D}$ est définie comme suit :*

$\sigma(g) = max\ \Sigma(g)$

alors que $\Sigma(g) = \{g' \in \mathcal{L}_i : g \subseteq_{sgi} g'$ *et* $\mathcal{D}[g] = \mathcal{D}[g']\}$

À partir de cette définition, nous remarquons facilement que la fermeture d'un graphe connexe g au sein de $\mathcal{L}_i$ consiste en l'ensemble des sous-graphes connexes de $\mathcal{L}_i$ maximaux ayant g comme prédécesseurs, et les mêmes occurrences que ce dernier au niveau de $\mathcal{D}$. Par conséquent, en commençant à partir d'un graphe générateur inductif g nous obtiendrons les sous-graphes fermés en exerçant des augmentations d'une arête à chaque fois jusqu'à ce que les sous-graphes générés auront une baisse de fréquence. Cependant, résoudre ce problème de recherche des maximaux revient à faire un nombre important de tests d'isomorphismes de sous-graphes.

Par ailleurs, en accord avec les deux représentations des graphes adoptés (discuté au niveau de la sous section 4.1), nous calculons ces maximaux par l'application de deux manières différentes, la recherche de l'ensemble des sous-graphes fermés successeurs d'un sous-graphe inductif est donnée comme suit :

4.4.1 Représentation par tableaux d'adjacences

Afin de récupérer les sous-graphes successeurs fermés à partir d'un sous-graphe générateur inductif X, un algorithme de recherche en profondeur d'abord est appliqué. Cet algorithme de recherche ajoute en boucle des arêtes au sous-graphe X en respectant la contrainte de ne pas perdre de fréquence. Ces arêtes $\mathcal{A}$ à ajouter sont le résultat de l'intersection des ensembles des clés des maps représentent les graphes G_i ayant X comme sous-graphes isomorphes.

Pour atteindre les sous-graphes fermés maximaux à partir de X, l'ensemble des sous-ensembles W d'arêtes de $\mathcal{A}$ est parcouru afin de récupérer tous les sous-graphes $X' = X \cup W$ tels que $\mathcal{D}[X] = \mathcal{D}[X']$ et les sous-ensembles W sont maximaux. Par conséquent, nous remarquons que pour le meilleur des cas, pour lequel il n'existe qu'un unique sous-graphe fermé Y à récupéré, il faut au moins $|E(Y) \backslash E(X)|$ tests de sous-graphes isomorphes, ainsi que des opérations de générations. Mais, comme le test d'isomorphisme de sous-graphe à base de comparaison des maps est peu coûteux dans le meilleur et le moyen des cas, alors en pratique comme le confirment les expérimentations que nous allons présenter **ADARCSgl** montre de très bonnes performances.

4.4.2 Représentation ensembliste

Comme nous l'avons déjà noté, le choix de la représentation ensembliste est motivé principalement par la simplicité de basculer vers le codage d'arêtes et d'arêtes converses (ECE) présenté au niveau de la sous section 3.3.2. La représentation ensembliste permet de simplifier les opérations d'augmentations et de tests de connexité, alors que la représentation ECE permet de réduire la complexité de l'isomorphisme de graphes et de sous-graphes à un test d'équivalence d'ensembles et un test d'inclusion d'ensembles respectivement.

La fermeture d'un graphe g consiste en l'intersection des codes ECE des graphes $G_i \in \mathcal{D}$ pour lesquels le code de g est inclus (c.-à-d. $\cap\mathcal{D}[ECE(G_i)]$ t.q $ECE(g)(G_i)$). Bien que ces nombreux avantages liés à l'encodage ECE. Le code (c.-à-d. le sous-ensemble d'items) issu du calcul de la fermeture peut correspondre à un sous-graphe non connexe et/ou ambigu. Ces deux derniers cas ont été détaillés sur la sous section 3.3.2. Par rapport au premier cas, pour un code d'un sous-graphe générateur inductif X l'idée utilisée est de chercher toutes les parties connexes du code, et de garder que celles qui incluent le code de X (c.-à-d. $ECE(X)$). Alors que pour le deuxième cas, les sous-codes de tailles deux des codes des triplets ambigus sont tous récupérés. Par la suite, chacun des codes

de triplets ambigus est remplacé un par un, par ses sous-codes de tailles deux. Enfin à partir de l'ensemble de sous-codes générés, que ceux incluent $ECE(X)$ sont gardés.

4.5 Description des algorithmes proposés

Au niveau de cette section, nous proposons deux algorithmes de découverts de sous-graphes connexes fréquents fermés à partir d'une base de graphes $\mathcal{D}$ à étiquetage d'arêtes non redondant. Ces deux algorithmes investissent l'accessibilité forte de l'espace de recherche de chacun des graphes $G_i \in \mathcal{D}$.

En commençant à partir du niveau des fermés C_0 qui contient uniquement le sous-graphe vide (c.-à-d. $\emptyset$) considéré toujours comme fermé fréquent tant que la base des graphes $\mathcal{D}$ est non redondante [Boley *et al.*2010], pour chaque niveau des fermés C_i chacun des sous-graphes fermés c_j est augmenté pour générer tous les sous-graphes générateurs inductifs possibles, pour chaque niveau des fermés C_i le niveau des sous-graphes générateurs inductifs générés est noté IN_i. Ensuite, en calculant la fermeture pour chacun des sous-graphes de IN_i, nous atteindrons le niveau prochain des fermés C_{i+1}. Afin d'éviter la redondance, tous les sous-graphes fermés et inductifs déjà atteints à des niveaux précédents sont éliminés. Enfin, ces étapes sont répétées en boucle jusqu'à ce qu'aucun sous-graphe générateur inductif n'est généré (c.-à-d. $IN_i = \emptyset$). L'algorithme 7 illustre le pseudo-code du schématique d'exploration adoptée.

Input: un graphe $G_i \in \mathcal{D}$, et un opérateur de fermeture σ.
Output: L'ensemble des sous-graphes fréquents fermés d'un graphe G_i $\sigma(\mathcal{F}_i)$.

E_F = Récupérer les arêtes fréquents de G_i;
$C_0 = \emptyset$;
$\sigma(\mathcal{F}_i) = C_0$;
IN_i =générerlesinduictifs($C_0,\ E_F,\ G_i$);
while *($IN_i! = \emptyset$)* **do**
  $C_{i+1} = calculerlafermeture(IN_i,\ E_F\ G_i)$;
  $\sigma(\mathcal{F}_i) = \sigma(\mathcal{F}_i) \cup C_{i+1}$;
  IN_i =générerlesinduictifs($C_{i+1},\ E_F,\ G_i$);
end
return $\sigma(\mathcal{F}_i)$;

Algorithm 7: | L'exploration de l'espace de recherche

Alors que les deux algorithmes que nous allons proposer ont le même principe pour

l'exploration de l'espace de recherche, ils se distinguent dans la manière de génération des sous-graphes générateurs inductifs et de calculer la fermeture de chacun de ces derniers.

4.5.1 ECERCSgl

Pour **ECERCSgl**, nous adoptons une représentation ensembliste des graphes. Cette représentation est utilisée pour les opérations d'augmentation et de test de la connexité, alors qu'il utilise une représentation d'arêtes et d'arêtes converses pour tester la redondance, la fréquence et pour calculer la fermeture.

L'algorithme **ECERCSgl** se comporte comme suit : 1) D'abord la base de graphes $\mathcal{D}$ est encodée comme une base transactionnelle d'itemsets $\mathcal{D}'$. 2) Ensuite, en commençant avec un ensemble de fermés C_i qui contient uniquement un sous-graphe $\emptyset$, l'ensemble des codes des sous-graphes générateurs inductifs est généré à partir de C_0. L'algorithme 8 illustre le pseudocode de la fonction de génération des sous-graphes générateurs inductifs. La fonction générerlesinduictifs() reçoit en entrée un ensemble de graphes fermés C_i, l'ensemble des arêtes fréquentes E_F et le graphe G_i, et génère l'ensemble des codes des graphes générateurs inductifs IN. Pour chacun des sous-graphes fermés $c \in C_i$ toutes les augmentations possibles qui génèrent des sous-graphes connexes sont appliquées. Pour chacune des augmentations maintenues g, le graphe est encodé comme un ensemble d'arêtes et d'arêtes converses $ECE(g)$. Ce dernier code est utilisé par la suite pour tester la fréquence par rapport à $\mathcal{D}'$. Si $ECE(g)$ est fréquent alors il est ajouté à l'ensemble IN.

Input: Un ensemble de graphes C_i, Un ensemble d'arêtes E_F, un graphe $G_i \in \mathcal{D}$.
Output: L'ensemble des codes ECE des sous-graphes générateurs inductifs IN.

```
IN = ∅;
for each graphe c : C_i do
    for each graphe e : E_F do
        if c ∪ e est connexe then
            gp=augmentation(c,e);
            ECE(gp) = encoder(gp);
            if frequent(ECE(gp), D') then
                IN = IN ∪ ECE(gp);
            end
        end
    end
end
return IN;
```

Algorithm 8: | La fonction générerlesinduictifs() (**ECERCSgl**)

3) Par la suite, pour chaque ensemble de codes de graphes IN_i non vide, nous calculons la fermeture. L'algorithme 9 illustre le pseudocode de la fonction de calcul de la fermeture des sous-graphes générateurs inductifs IN_i. Cette dernière fonction reçoit en entrée un ensemble de codes ECE de graphes générateurs inductifs IN_i, l'ensemble des arêtes fréquentes E_F et le graphe G_i, et génère l'ensemble des graphes fermés C_{i+1}. Pour chacun des codes ECE, $ECE(x) \in IN_i$ l'intersection de tous les codes de $\mathcal{D}'$ qui incluent le code $ECE(x)$ sans l'existence de faille de codage (présentée au niveau de la sous section 4.2.1). Alors pour chacun des codes $ECE(x)$ un nouveau code $ECE(y)$ est généré tel que $ECE(y) = \cap\mathcal{D}'[ECE(x)]$. Ce dernier code peut souffrir de deux problèmes : a) l'ambiguïté, et b) la non-connexité. Alors chaque nouveau code généré $ECE(x)$ est testé par rapport à l'ambiguïté. Si ce dernier est ambigu le code $ECE(x)$ sera remplacé par un ensemble de sous-codes non ambigus $MaxECE$, pour chacun de ces codes $MaxECE$ un graphe g'' est reconstruit. Si le code est non ambigu, alors le graphe g'' correspondant est reconstruit. Pour le problème de connexité, il est résolu au niveau de la reconstruction en ignorant la partie non connexe. Enfin chacun des graphes reconstruits est ajouté à l'ensemble des sous-graphes fermés C_i.

4) Pour chaque nouvel ensemble de sous-graphes fermés généré C_i, ils sont ajoutés à l'ensemble des fermés $\sigma(\mathcal{F}_i)$. Les deux étapes 2 et 3 sont répétées en boucle jusqu'à ce qu'aucun nouveau sous-graphe générateur inductif ne soit pas généré et enfin l'ensemble des fermés $\sigma(\mathcal{F}_i)$ est retourné.

4.5.2 ADARCSgl

Comme nous l'avons déjà discuté, **ADARCSgl** partage la même idée que **ECERCSgl** vis-à-vis de l'exploration de l'espace de recherche. Mais il se comporte différemment pour tester la fréquence et pour calculer la fermeture.

Alors afin de procéder à des augmentations, la fonction générerlesinduictifs() au niveau de l'algorithme 10, génère l'ensemble des graphes générateurs inductifs IN comme pour **ECERCSgl** sauf que l'opération de test à ce niveau consiste à tester l'isomorphisme de sous-graphes d'un graphe gp augmenté à partir d'un graphe fermé c par rapport à tous les sous-graphes de $\mathcal{D}$ qui contient c. Malgré cette dernière optimisation par rapport au fait de tester l'isomorphisme de sous-graphes pour tous les graphes de $\mathcal{D}$, mais la complexité du test au pire des cas complique considérablement cette tâche de génération des sous-graphes générateurs inductifs.

Input: Un ensemble des codes de graphes IN_i, Un ensemble d'arêtes E_F, un graphe $G_i \in \mathcal{D}$.
Output: L'ensemble des sous-graphes fermés C_i.

for *each graphe* $ECE(x) : IN_i$ **do**
  $ECE(y) = \cap\mathcal{D}'[ECE(x)]$;
  if $ECE(y)$ *est ambigu* **then**
    $MaxECE$= récupérersouscodes(ECE(y));
    for *each code* $ECE(z) : MaxECE$ **do**
      $g'' = reconstruire(ECE(z), G_i)$;
      $C_i = C_i \cup g''$;
    end
  end
  else
    $g'' = reconstruire(ECE(y), G_i)$;
    $C_i = C_i \cup g''$;
  end
end
return C_i;

Algorithm 9: | La fonction calculerlafermeture() (**ECERCSgl**)

Input: Un ensemble de graphes C_i, Un ensemble d'arêtes E_F, un graphe $G_i \in \mathcal{D}$.
Output: L'ensemble des sous-graphes générateurs inductifs IN.

$IN = \emptyset$;
for *each graphe* $c : C_i$ **do**
  for *each graphe* $e : E_F$ **do**
    if $c \cup e$ *est connexe* **then**
      gp=augmentation(c,e);
      if $frequent(gp,\ \mathcal{D}[c])$ **then**
        $IN = IN \cup gp$;
      end
    end
  end
end
return IN;

Algorithm 10: | La fonction générerlesinduictifs() (**ADARCSgl**)

Pour récupérer les sous-graphes successeurs fermés à partir de chacun des sous-graphes générateurs inductifs x, la fonction calculerlafermeture() au niveau de l'algorithme 11, applique un algorithme de recherche en profondeur d'abord getfermeture() pour atteindre l'ensemble des sous-graphes fermés Y t.q $\forall y \in Y\ \mathcal{D}[x] = \mathcal{D}[y]$. Cet algorithme de recherche

Input: Un ensemble de graphes IN_i, un graphe $G_i \in \mathcal{D}$.
Output: L'ensemble des sous-graphes fermés C_i.

for *each graphe* $x : IN_i$ **do**
  $Y = getfermeture(x, G_i)$;
  $C_i = C_i \cup Y$;
end
return C_i;

Algorithm 11: | La fonction getfermeture() (**ADARCSgl**)

ajoute en boucle des arêtes au sous-graphe X en respectant la contrainte de rester connexe au niveau de G_i et de ne pas perdre de fréquence par rapport $\mathcal{D}[x]$. Nous remarquons que pour le meilleur des cas, pour lequel il n'existe qu'un unique sous-graphe fermé y à récupérer, il faut au moins $|E(y) \backslash E(x)|$ tests de sous-graphes isomorphes, ainsi que d'opérations d'augmentations.

4.6 Résultats expérimentaux

Dans cette section, nous présentons des expérimentations menées en utilisant des données synthétiques. Ces expérimentations consistent à tester les performances des algorithmes proposés (c.-à-d. **ECERCSgl**, **ADARCSgl**) par rapport à un ensemble de base de graphes synthétiques. Nous notons que les deux algorithmes proposés ont été implémentés avec le langage Java, et pour avoir des résultats représentatifs nous avons utilisé deux générateurs de base de graphes différents : 1) GraphGen [Cheng *et al.*2006], et 2) GenUniqueEdgesLabelsGraph.

Pour le premier générateur, principalement 5 paramètres sont considérés lors de la génération. Deplus que la taille T espérée pour la base, S la taille des graphes (c.-à-d. le nombre d'arêtes), ce paramètre a été défini comme une distribution d'une loi normale avec la moyenne comme entrée et 5 pour la variance. ETE, ETV le nombre d'étiquettes d'arêtes et de sommets, respectivement. L représente la densité des graphes, entre [0,1] le graphe est complet si $L = 1$, alors il est totalement sparse pour $L = 0$. Par rapport à cette étude expérimentale, nous avons généré des bases avec des tailles différentes T de 100 à 1000 graphes. Les paramètres appliqués lors de la génération des bases de graphes sont : 1) S prend les valeurs 20, 30, et 50, ce qui permet d'avoir avec une probabilité élevée des tailles de graphes entre $[5, 65]$. 2) Le nombre d'étiquettes des sommets et des arêtes : a) 10, et 20 pour les sommets V, et b) fixé à 20 pour les arêtes. 3) La densité des graphes générés prend comme valeurs 0.1, 0.4, 0.7 et 0.8. Ces variations des paramètres

ont comme but de générer des cas de base de graphes différents, afin d'avoir une meilleure image des comportements des tests effectués.

Malheureusement, les bases de graphes générées en utilisant GraphGen ne satisfont pas la contrainte d'étiquettes d'arêtes non redondante et que ce dernier ne permet pas de générer des bases ayant des graphes particuliers. Pour satisfaire la contrainte d'étiquetage d'arêtes non redondant, pour chacune des bases générées en utilisant GraphGen, une phase de prétraitement est utilisée, pour surétiqueter les graphes afin de satisfaire la contrainte d'étiquetage d'arêtes non redondant. Mais cette phase de prétraitement déstabilise les configurations souhaitées par rapport aux densités fixées. Alors pour avoir des bases de graphes particuliers ayant un étiquetage unique pour les sommets ou une particularité de générer des graphes complets, nous avons utilisé notre générateur de graphes GenUniqueEdgesLabelsGraph.

Comme nous l'avons déjà mentionné au niveau de la sous section 4.1.3, GenUniqueEdgesLabelsGraph est un générateur de base de graphes à étiquetage d'arêtes non redondant. Nous avons considéré 5 paramètres, S la taille des graphes (c.-à-d. le nombre d'arêtes), ce paramètre a été défini comme une fonction ayant une image aléatoire entre $[x-5$, $x+5]$ de la taille en entrée x. ETE, ETV le nombre d'étiquettes d'arêtes et de sommets, respectivement. Enfin L représente la densité des graphes, entre 0, 0.5, et 1 le graphe est complet si $L=1$, alors il est totalement sparse pour $L=0$. Pour la présente expérimentation, nous avons généré des bases de tailles T de 100 graphes. Les paramètres appliqués lors de la génération des bases de graphes sont : 1) S prend les valeurs 20, 30, et 50, ce qui permet d'avoir des tailles de graphes entre $[5, 55]$. 2) Le nombre des étiquettes des sommets et des arêtes : a) 1, 2, et 5 pour les sommets V, et b) de 20 à 100 pour les arêtes. 3) La densité des graphes générés prend comme valeurs 0.1, 0.5, et 1.0. Ces variations des paramètres ont comme but de générer des cas de base de graphes particuliers (c.-à-d. graphes complets, étiquetage de sommets unique, etc.), afin d'avoir une meilleure image des comportements des deux algorithmes proposés par rapport à des cas particuliers.

D'abord, nous commençons cette étude expérimentale du comportement des algorithmes proposés en utilisant des bases de graphes générés avec GraphGen, avec une variation du nombre de graphes des bases de graphes entre 100, et 500 et un seuil de fréquence de 2%, la table 4.6 présente : 1) Le nombre total des sous-graphes connexes fréquents.L'ensemble de ces sous-graphes connexes fréquent récupéré par le biais d'un algorithme qui explore tout l'espace de recherche et énumère les sous-graphes fréquents. 2) Le nombre des sous-graphes connexes fréquents fermés énumérés par **ECERCSgl** et

ADARCSgl, ainsi que les temps d'exécutions de chacun d'entre eux. 3) Le temps d'exécution et le nombre de sous-graphes connexes listés par un algorithme de découverte de sous-graphes connexes fréquents à étiquetage d'arêtes non redondant qui explore tous l'espace de recherche pour récupérer tous les fréquents et ensuite filtre ces derniers pour ne garder que les fermés parmi eux. Pour cette expérience les paramètres utilisés sont $S = 20$ et $S = 30$, $ETV = 10$, $ETE = 20$ et une densité L qui couvre toutes les possibilités (c.-à-d. 0.1, 0.4, et 0.7).

TABLE 4.6 – Les résultats à un seuil de 2% des bases de graphes pour $S = 20$ et $S = 30$.

Bases de graphes					**ECERCSgl**	**ADARCSgl**	$\mathcal{NAIF}$
$\|\mathcal{D}\|$	S	Id	$\|\mathcal{F}\|$	$\|\mathcal{C}\|$	$temps(sec)$	$temps(sec)$	$temps(sec)$
	20	ETV20ETE20D0.1	2867	1420	1.765	2.214	10.321
$T = 100$	20	ETV20ETE20D0.4	5032	1660	4.868	4.274	20.594
	20	ETV20ETE20D0.7	6058	1059	12.44	7.377	28.31
	20	ETV20ETE20D0.1	4961	2004	5.43	6.296	62.007
$T = 200$	20	ETV20ETE20D0.4	6342	2207	9.056	6.494	43.978
	20	ETV20ETE20D0.7	7224	1897	15.081	9.401	63.381
	30	ETV20ETE40D0.1	5690	3364	29.376	25.26	199.862
$T = 300$	30	ETV20ETE40D0.4	7351	4144	60.874	30.459	338.927
	30	ETV20ETE40D0.7	7731	3954	124.122	50.849	439.607
	30	ETV20ETE40D0.1	5692	3590	77.757	74.799	1129.301
$T = 400$	30	ETV20ETE40D0.4	8395	5585	183.298	94.859	1199.694
	30	ETV20ETE40D0.7	8909	5149	440.352	178.312	1291.047
	30	ETV20ETE40D0.1	6045	4243	153.051	181.675	3488.831
$T = 500$	30	ETV20ETE40D0.4	9234	6721	487.61	202.924	2481.303
	30	ETV20ETE40D0.7	9693	5982	944.77	478.803	3591.94

La table 4.6 illustre l'importante différence entre la taille des sous-graphes fréquents $\mathcal{F}$ et celle des fréquents fermés $\mathcal{C}$. La figure 4.5 illustre une représentation graphique de ces derniers résultats, la partie (a) représente les taux des sous-graphes fréquents $\mathcal{F}$ et celle des fréquents fermés $\mathcal{C}$ pour une densité de 0.1, (b) représente les taux pour une densité de 0.4 et (c) pour une densité de 0.7.

Cette table 4.6 illustre aussi une totale domination des algorithmes proposés par rapport à l'algorithme $\mathcal{NAIF}$, où **ECERCSgl** et **ECERCSgl** consomment des temps d'exécutions de l'ordre de 1/10 par rapport à ce dernier. Nous remarquons aussi que les résultats **ADARCSgl** sont largement meilleurs que ceux de **ECERCSgl**, où autres que de faire

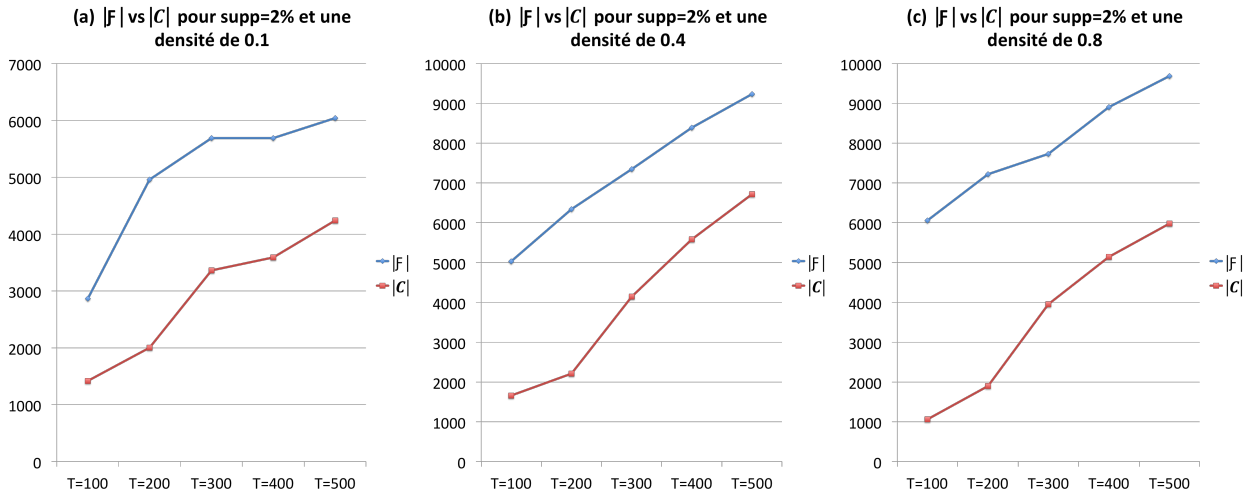

FIGURE 4.5 – Représentation graphique des résultats $\mathcal{F}$ vs $\mathcal{C}$

face à des graphes ayant de petites tailles (c.-à-d. $S = 10$ ou 20) et que ces derniers soient largement sparses (c.-à-d. $L = 0.1$), les temps d'exécutions de **ADARCSgl** sont beaucoup plus efficaces. Encore, nous remarquons facilement qu'en plus de la taille des bases de graphes T, les augmentations de densités nécessitent des temps d'exécutions plus importants soit pour **ECERCSgl** ou **ADARCSgl**. Cependant, pour **ECERCSgl** les augmentations des temps sont beaucoup plus importantes vis-à-vis du fait que l'augmentation de densité augmente le nombre de triplets au sein d'un graphe ce qui complique davantage les opérations de codages et de décodages ainsi que les tests d'inclusions et d'équivalence des codes. La figure 4.6 illustre une représentation graphique des temps d'exécution de **ECERCSgl** vs **ADARCSgl** par rapport aux différentes variantes de la densité.

Afin d'avoir une image plus pertinente des performances des deux algorithmes proposés, et de confirmer l'impact de l'augmentation des tailles des graphes par rapport à l'efficacité des algorithmes, en utilisant des bases de graphes générés avec GraphGen, avec une variation du nombre de graphes des bases de graphes entre 100, et 800 et un seuil de fréquence de 2%, la table 4.7 présente : 1) Le nombre total des sous-graphes connexes fréquents. 2) Le nombre des sous-graphes connexes fréquents fermés énumérés par **ECERCSgl** et **ADARCSgl**, ainsi que les temps d'exécution de chacun d'entre eux. Pour cette expérience les tailles des graphes sont fixées à $S = 50$, ETV à 10 et 20, ETE entre 10 et 100 et une densité D qui couvre toutes les possibilités (c.-à-d. 0.1, 0.4, et 0.8).

La table 4.7 illustre un surclassement en terme de performance de **ADARCSgl** ce qui est dû à l'augmentation des tailles des graphes qui affectent la phase de codage qui néces-

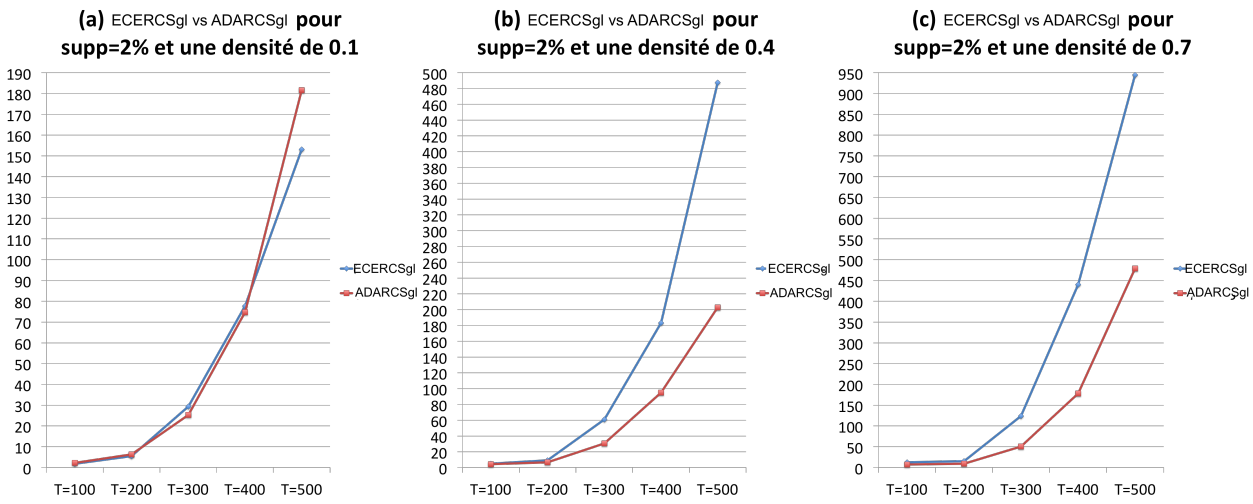

FIGURE 4.6 – Temps d'exécutions de **ECERCSgl** vs **ADARCSgl** (1).

TABLE 4.7 – **ECERCSgl** vs **ADARCSgl** pour des bases de graphes $S = 50$.

Bases de graphes				**ECERCSgl**	**ADARCSgl**
$\|\mathcal{D}\|$	*Id*	$\|\mathcal{F}\|$	$\|\mathcal{C}\|$	*temps(sec)*	*temps(sec)*
	S50ETV20ETE10D0.1	3905	2489	3.668	3.616
T=100	S50ETV20ETE10D0.4	6807	2974	14.421	12.923
	S50ETV20ETE10D0.6	8457	2545	31.345	31.387
	S50ETV20ETE20D0.1	4872	2367	9.162	6.131
T=200	S50ETV20ETE20D0.4	8208	3222	26.8	14.36
	S50ETV20ETE20D0.8	9182	3092	44.67	23.168
	S50ETV20ETE50D0.1	5897	4616	164.772	82.179
T=500	S50ETV20ETE50D0.4	9525	7297	444.382	193.737
	S50ETV20ETE50D0.8	11711	8208	725.838	401.226
	S50ETV20ETE100D0.1	6583	5766	641.398	257.666
T=800	S50ETV20ETE100D0.4	12773	10936	2256.483	856.2971
	S50ETV20ETE100D0.8	14539	11949	1853.781	1564.095

site des temps d'exécutions encore plus importants. Les temps d'exécutions démontrent un large avantage qui peut aller jusqu'à 1/3 pour **ADARCSgl**. La figure 4.7 illustre une représentation graphique des temps d'exécutions de **ECERCSgl** vs **ADARCSgl** présenté au niveau de la table 4.7 par rapport aux différentes variantes de densité.

Mais contrairement à ces cas généraux qui ont permis à **ADARCSgl** de prendre l'avantage, comme nous l'avons discuté au niveau de la sous section 4.2.3, il existe bien

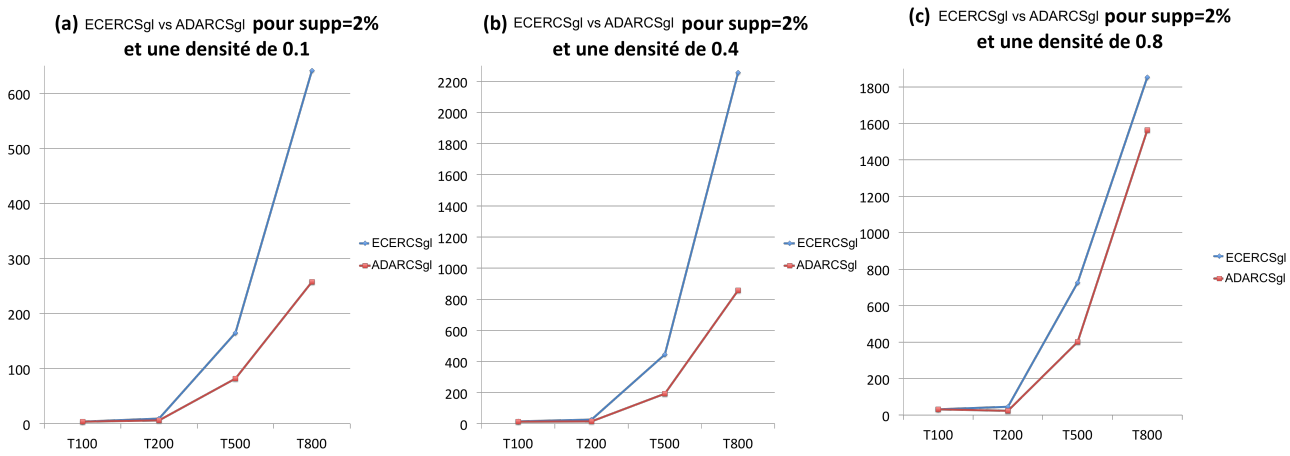

FIGURE 4.7 – Temps d'exécutions de **ECERCSgl** vs **ADARCSgl** (2).

quelques cas particuliers qui passent en faveur de **ECERCSgl** et son encodage ECE. Pour avoir une idée par rapport au comportement des algorithmes proposés, nous menons un ensemble d'expérimentations par rapport aux bases de données générées avec notre générateur de graphes GenUniqueEdgesLabelsGraph, ce qui nous permettra de générer des graphes complètement sparses (c.-à-d. $L = 0$) ou complets (c.-à-d. $L = 1$) et même à étiquetage des sommets unique. Cette expérimentation a comme objective d'illustrer le comportement des deux algorithmes par rapport au pire des cas de **ADARCSgl**. Pour ce cas présent, le test d'isomorphisme de sous-graphe basé sur la comparaison des tableaux d'adjacences, est d'un ordre exponentiel **ADARCSgl** (c.-à-d. $|V| = 1 \leftrightarrow \mu = |E|$, donc le test est de complexité $\mathcal{O}(2^{|E|})$). La Table 4.8 liste les résultats issus contre des bases de graphes de tailles 100 par rapport à un seuil de 2%. Les autres paramètres ont été variés comme suit : S à comme valeurs 20, 30, 50, et 70, ETV pour des valeurs de 1, 2, et 5, ETE à comme valeurs $|S| + 20$, et enfin une densité L pour 0 et 1.

Les résultats au niveau de la table 4.8 peuvent être interprétés par rapport à deux points : 1) l'étiquetage des sommets, et 2) la densité des graphes. Nous remarquons par rapport à la réduction du nombre d'étiquettes des sommets des graphes que les résultats confirment l'avantage attendu de **ECERCSgl** à l'égard de **ADARCSgl**. Pour le pire des cas de **ADARCSgl** où pour le cas de l'étiquetage des sommets est unique (c.-à-d. ETV=1), les résultats illustrent un large avantage des temps d'exécutions de **ECERCSgl**. Cependant, de larges augmentations du nombre d'étiquettes des sommets réduisent l'avantage de **ECERCSgl** et que ce dernier avantage est même basculé en faveur de **ADARCSgl**. Pour un nombre d'étiquettes autre que $ETV = 1$ les résultats sont comparatives à ceux présentés au niveau des tables 4.6 et 4.7, où la taille des graphes et

la densité sont les principales clés qui influencent le comportement des deux algorithmes.

TABLE 4.8 – **ECERCSgl** vs **ADARCSgl** pour le cas d'un étiquetage de sommets faible.

Bases de graphes		**ECERCSgl**	**ADARCSgl**
Id	$\|\mathcal{C}\|$	*temps*(*sec*)	*temps*(*sec*)
T100S20ETv1ETE40D0.0	2202	1.963	3.0
T100S20ETv1ETE40D1.0	10305	25.185	62.806
T100S20ETv2ETE40D0.0	1082	0.91	1.533
T100S20ETv2ETE40D1.0	2594	3.755	4.169
T100S20ETv5ETE40D0.0	566	0.615	0.868
T100S20ETv5ETE40D1.0	688	0.993	0.894
T100S30ETv1ETE50D0.0	3467	4.04	6.478
T100S30ETv1ETE50D1.0	32282	312.891	1386.921
T100S30ETv5ETE50D0.0	809	0.801	1.167
T100S30ETv5ETE50D1.0	1192	2.698	1.915
T100S50ETv2ETE70D0.0	2869	2.87	3.121
T100S50ETv2ETE70D1.0	16035	311.685	519.13
T100S50ETv5ETE70D0.0	1234	1.632	1.723
T100S50ETv5ETE70D1.0	2407	11.037	8.367
T100S70ETv2ETE90D0.0	4103	4.698	5.012
T100S70ETv2ETE90D1.0	32614	3942.376	8824.854
T100S70ETv5ETE90D0.0	1591	2.268	1.81
T100S70ETv5ETE90D1.0	4194	49.869	29.46

Conclusion du chapitre

Nous avons présenté dans ce chapitre le problème de la découverte des sous-graphes fermés connexes fréquents pour la classe des graphes à étiquetage d'arêtes non redondant. Les approches existantes ont été proposées pour le cas général, ces derniers utilisent différentes techniques afin de réduire l'espace de recherche (le nombre de motifs à visiter pour atteindre les fermés et les maximaux), ce qui réduira par la suite le nombre de tests des sous-graphes isomorphes. Malgré ces techniques d'optimisation, aucun algorithme n'a pu atteindre la performance de ceux développés pour le cas des itemsets, des ensembles, des clés ...etc. Principalement, ce manque de performance pour les graphes est dû à la complexité de l'espace de recherche et le coût élevé des tests de fréquence.

Alors deux approches efficaces **ECERCSgl** et **ADARCSgl** pour la découverte des sous-graphes connexes fréquents fermés à étiquetage d'arêtes non redondant ont été propo-

sées. Les points forts de ces approches sont la réduction du nombre de graphes parcourus avant d'atteindre tous les fermés, ainsi que l'utilisation de la relation d'ensemble (sous ensemble d'un ensemble) et une comparaison de tableaux pour le test de fréquence et le calcul de la fermeture.

Chapitre 5

La construction d'un arbre de décision a base de contraintes

Sommaire

DANS ce chapitre, une nouvelle approche de construction d'arbres de décisions est présentée. Cette nouvelle technique permet de résoudre les problèmes de sur-apprentissage et de complexité liés à la phase de construction. La phase de construction d'un arbre de décision souffre des problèmes de sélection d'attributs et d'exemples d'apprentissage. Alors, comme solution, une combinaison entre un processus de sélection d'attributs et un échantillonnage des données a été proposée [Karabadji *et al.*2014].

5.1 La construction d'arbres de décisions

Naturellement, la construction d'un classificateur se déroule principalement en deux étapes : d'abord une phase d'apprentissage, effectuée avec un échantillon d'exemples (sous-ensemble de la base d'exemples). Ensuite, une étape de test qui permet de vérifier les performances du classificateur produit à l'égard d'un nouvel échantillon. Cette dernière étape permet de sélectionner le meilleur des modèles élaborés dans l'étape d'apprentissage. Pour la construction d'un arbre de décision, les phases d'apprentissage et celle du test sont en chevauchement. La construction d'un arbre suit deux étapes principales, l'augmentation

et l'élagage.

Par rapport à l'étape au l'arbre AR est poussé, les exemples d'apprentissage sont divisés à deux, ou plusieurs sous-ensembles descendants, sachant que la division d'exemples est exécutée en respectant un critère de séparation. Ce processus est répété en boucle jusqu'à ce qu'aucune nouvelle division ne soit possible, ou qu'aucune condition d'arrêt n'est atteinte. Cette étape génère pour la plupart des cas un arbre de décision très complexe qui souffre du sur-apprentissage dû à la considération des incertitudes des exemples d'apprentissages (c.-à-d. des particularités, du bruit, et les duplicatas).

Cependant, afin de contourner ces deux derniers phénomènes bloquants, beaucoup d'approches d'élagages ont été proposées. Ces techniques d'élagage sont des heuristiques qui permettent de résoudre le problème de sur-apprentissage, par l'élagage de toutes les sections de l'arbre qui sont construites à partir de données souffrent de problèmes de bruit et/ou sont des données erronées. En plus, les sections élaguées de l'arbre permettent la réduction de la complexité et la taille de l'arbre.

Dans la littérature, beaucoup de techniques d'élagage ont été présentées. Ces techniques partagent les mêmes objectifs qui consistent en la réduction de : (1) la complexité de l'arbre, et (2) l'erreur de précision de classification par rapport à un ensemble d'exemples de test indépendant. Cependant, ces méthodes se distinguent sur pas mal de points, leurs différences sont catégorisées comme suit :

1. La nécessité d'un ensemble indépendant d'exemples de test ;
2. La génération d'une série d'arbres élagués ;
3. Le critère de décision d'élagage.

[Breiman *et al.*1984] ont développé la méthode d'élagage de la complexité d'erreur (error-complexity pruning). Cette technique utilise une mesure d'élagage appelé le risque dû au coût de la complexité (c.-à-d. cost-complexity risk), basé sur une pénalisation du taux d'erreur et de la taille du sous-arbre traité. Pour calculer cette mesure d'élagage, les erreurs de précisions de classifications, ainsi que la taille des feuilles des arbres de décisions sont considérées. Le risque du coût de complexité RC est calculé pour tous les sous-arbres possibles, en commençant par l'arbre de décision initial AR_0, ce coût de complexité est mesuré pour un sous-arbre t comme étant la somme d'erreur d'apprentissage $R(t)$ et du produit d'un facteur α et le nombre de feuilles du sous-arbre $|\bar{t}|$, c.-à-d. $RC_{\alpha(|\bar{t}|)} = R(t) + \alpha(|\bar{t}|)$. Ensuite, une série d'arbres de décisions élagués ayant les plus

petites valeurs de α est sélectionnée. Enfin, le meilleur arbre de décision qui affiche les performances les plus optimales, par rapport à la classification d'un ensemble d'exemples de tests indépendants, est sélectionné à partir de la série α d'arbres de décisions élagués.

Proposée dans [Quinlan1987], la technique de réduction d'erreur d'élagage (Reduced-error pruning) produit une série d'arbres élagués en utilisant un ensemble d'exemples de tests. À partir de l'arbre AR_0 qui a été poussé à base d'exemples d'apprentissage, pour chacun des noeuds t de AR_0 les taux d'erreurs de précisions de classifications des exemples de tests sont calculés : a) α le cas où le sous-arbre pour lequel t est la racine, est remplacé par une feuille, et b) β le cas où le sous-arbre ne sera pas élagué. Ensuite, pour chacun des noeuds t, en lui associant la différence de ses deux taux d'erreur (α, β). Le noeud avec la plus grande différence est donc sélectionné pour être élagué (c.-à-d. le sous-arbre remplacé par une feuille). Ce processus est répété jusqu'à ce que l'élagage augmente l'erreur de précision de classification de l'ensemble des exemples de tests. Cette méthode d'élagage permet de générer le plus petit arbre de décision qui généralise l'ensemble des exemples d'apprentissage, mais avec un grand risque de sur élagage de l'arbre.

L'erreur de base d'élagage (Error-based pruning, EBP) est une autre méthode proposée par Quinlan dans [Quinlan1993]. EBP est une version améliorée de la technique d'élagage d'erreur pessimistic [Quinlan1987] où elle n'utilise pas un ensemble indépendant d'exemples de tests. Cette technique explore l'arbre de décision AR_0 en profondeur, pour chacun des noeuds x de l'arbre, contrairement à d'autres techniques d'élagage qui remplacent un sous-arbre ayant un noeud x comme racine, par une feuille (classe), EBP permet la possibilité de greffer un sous-arbre t_y du sous-arbre t_x à la place du sous-arbre t_x. Pour pouvoir prendre une décision d'élagage d'un noeud x, EBP calcule l'estimation d'erreur pour chacun des noeuds. Les valeurs d'estimation d'erreurs permettent à EBP de déterminer si un sous-arbre est à remplacer par une feuille, l'un de ses sous-arbres, où de garder l'arbre originaire.

Enfin d'autres techniques d'élagage ont été proposées, nous citons la valeur critique d'élagage (critical value pruning) [Mingers1987], l'erreur minimum d'élagage (minimum error pruning) [Niblett and Bratko1987], DI élagage (Depth and the Impurity of nodes pruning) [Fournier and Crémilleux2002], et une méthode d'élagage basé sur le risque structurel des feuilles [Luo *et al.*2013]. Malgré ses avantages la phase d'élagage peut sur élaguer, ou sous-élaguer l'arbre de décision. En plus, aucune des nombreuses techniques ne surclasse les autres [Quinlan1987], [Quinlan1993], [Breiman *et al.*1984], [Niblett and Bratko1987]. De plus, en cherchant tous les sous-arbres (c.-à-d. sections) à élaguer, chaque noeud de

l'arbre doit être parcouru. Autrement dit, le processus d'élagage peut être illustré comme un processus de découverte d'un sous-arbre optimal AR^* au sein d'un espace de recherche composé de tous les sous-arbres de AR.

D'après la description du processus de construction d'un arbre de décision, ainsi que les problèmes liés à ses deux phases, nous pouvons résumer la phase d'augmentation comme une phase d'extraction d'une combinaison d'attributs. Alors que la phase d'élagage peut être présentée comme un processus de sélection du sous-arbre de décision le plus performant par rapport à un ensemble d'exemples indépendants de test. Cette sélection d'un sous-arbre par rapport à un ensemble d'exemples de test autre que celui utilisé pour l'apprentissage a comme objectif d'éliminer les sections construites à base des particularités, de bruit, ou/et des erreurs.

Dans la littérature, d'autres techniques ont prouvé leurs performances. Ces dernières permettent même de surpasser les problèmes de construction avant de commencer l'apprentissage. Ces techniques sont la sélection d'attributs et l'échantillonnage des données. Beaucoup d'études ont prouvé l'amélioration des performances des classificateurs construits en utilisant la sélection d'attributs, ou d'une sélection d'un sous-ensemble d'exemples d'apprentissage [Macaš *et al.*2012], [Piramuthu2008], [Ishibuchi *et al.*2001], [Bermejo *et al.*2012], [Liu2010]. Cependant, pour faire face au sur-apprentissage et la complexité de l'arbre simultanément nous proposons de combiner la sélection d'attributs et la réduction de l'échantillon d'apprentissages. L'algorithme $\mathcal{IUDT}$ (Improved Unpruned Decision Tree) proposé permet la construction d'un arbre de décision optimal sans faire appel au processus d'élagage. Alors la construction d'un arbre de décision est réduite comme étant un problème de découverte d'un couple d'un jeu d'attributs et un autre d'exemple d'apprentissage parmi toutes les combinaisons possibles permettent la construction d'un arbre de décision le plus générique possible sans faire appel à la phase d'élagage.

5.2 L'approche $\mathcal{IUDT}$

Cette section décrit l'idée de remplacer la phase post-élagage par la sélection d'attributs et de la réduction de données d'apprentissage. L'objectif principal est de démontrer que la performance d'un DT qui ne sera pas élagué peut être améliorée en utilisant ces deux étapes. La sélection d'attributs "wrapper" permet d'éliminer les attributs pertinents et redondants, et la réduction des données d'apprentissage permet de réduire le problème de la complexité de la taille. Nous devons donc déterminer à la fois le meilleur sous-ensemble d'attributs et le sous-ensemble d'exemples d'apprentissage utilisés pour

construire un arbre de décision robuste sans qu'il soit taillé. Les principales caractéristiques de la méthode proposée sont les suivants : (i) prétraitement des données, (ii) la définition des combinaisons des sous-ensembles d'attributs et d'exemples d'apprentissage comme un espace de recherche (graphe de spécialisation), et (iii) l'application d'une exploration en largeur d'abord à l'apriori sur l'espace de recherche pour trouver l'un des meilleurs arbres de décisions t^* non élagué. La figure 5.1 présente le schéma de la méthode proposée.

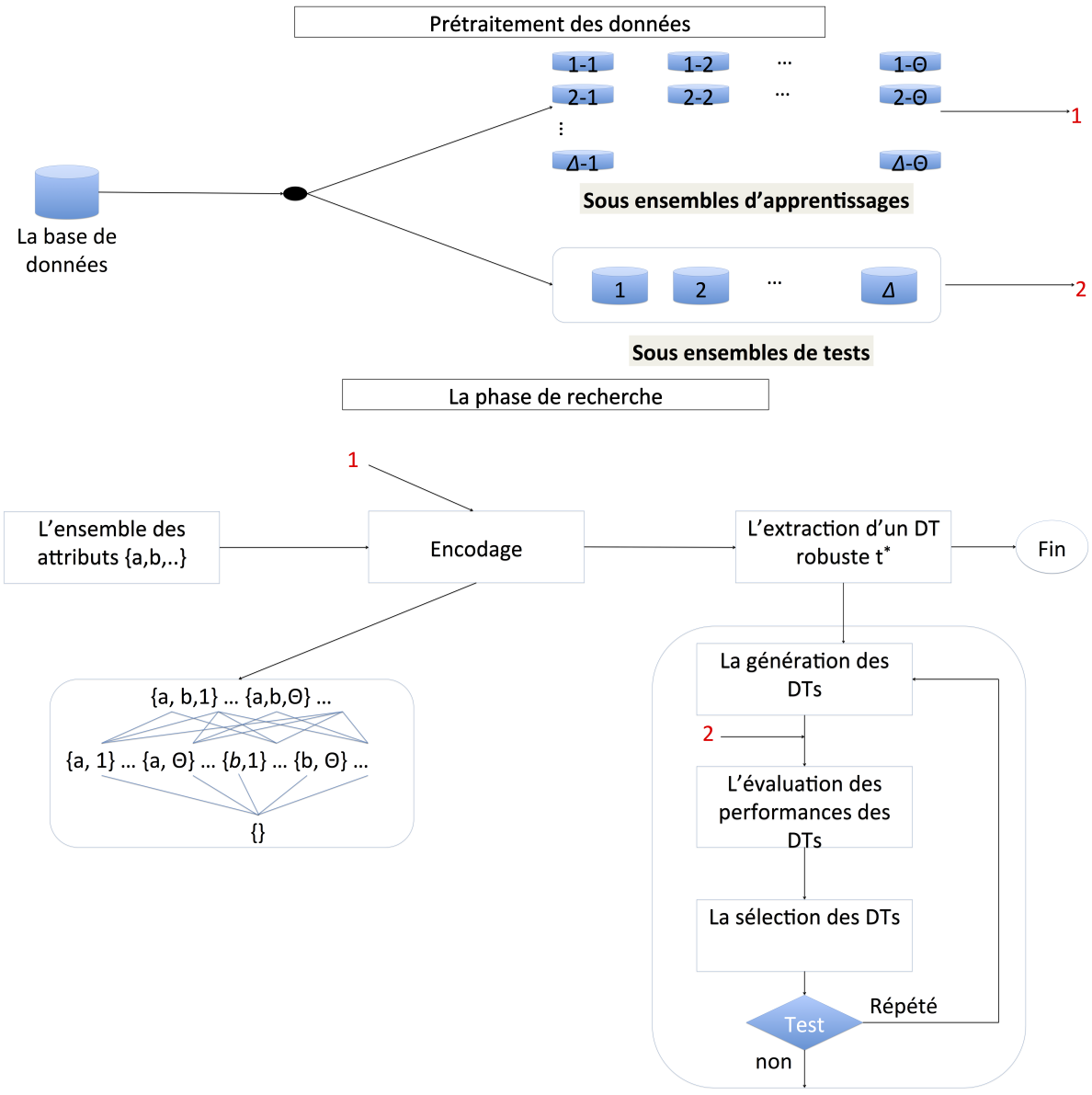

FIGURE 5.1 – L'approche $\mathcal{IUDT}$.

5.2.1 Prétraitement des données

Beaucoup de travaux d'analyses expérimentales sur les méthodes d'élagages ont utilisé un échantillonnage par division aléatoire en respectant les pourcentages : 25 coupes de 70% d'exemples d'apprentissage et 30% d'exemples de test [Esposito *et al.*1997], 9 coupes de 60% d'exemples d'apprentissage et 40% d'exemples de test [Mingers1989], et 10 coupes de 25% d'exemples d'apprentissage et 75% d'exemples de test [Bradford *et al.*1998]. Ces nombreuses coupes ont comme principal objectif d'éviter la non-représentativité des résultats issue d'un unique échantillonnage aléatoire.

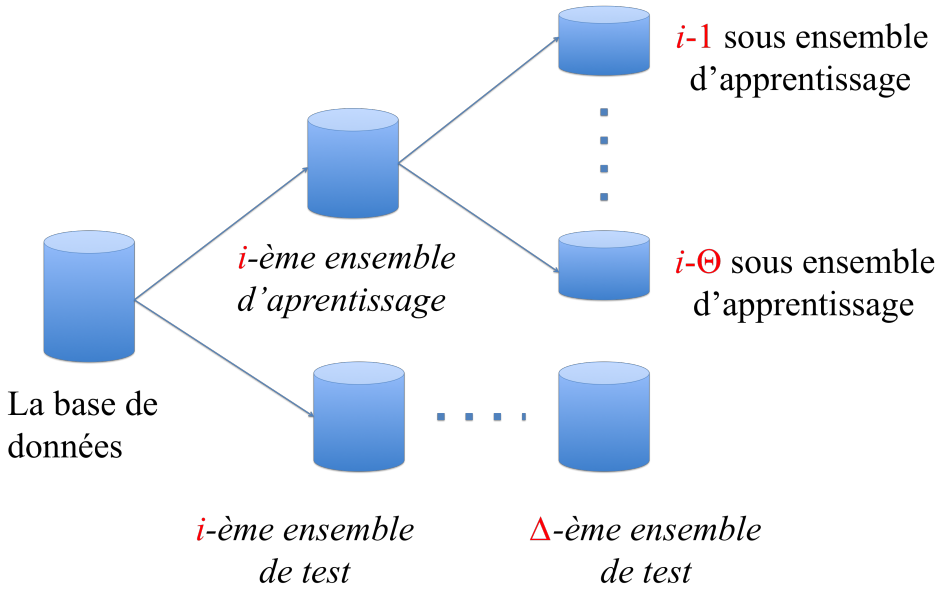

FIGURE 5.2 – Le processus d'échantillonnage.

Pour avoir un échantillonnage avec des résultats représentatifs, $\mathcal{IUDT}$ divise la base d'exemples en deux %50 pour l'apprentissage et %50 pour le test. Ce processus d'échantillonnage est répété Δ fois, chaque échantillon d'apprentissage est encore aléatoirement échantillonné en Θ sous-échantillons. Donc, nous allons avoir $(\Delta * \Theta)$ échantillons d'apprentissage différents $\mathcal{S}_{\mathcal{IT}}$, ainsi que Δ échantillons de tests $\mathcal{S}_{\mathcal{S}}$. Δ et Θ sont définis par les usagers. Les chevauchements et la sélection aléatoire des échantillons sont la garantie principale de validité de ce schéma d'échantillonnage afin d'avoir une solide estimation d'erreurs de classification. Par rapport aux expérimentations que nous avons réalisées, Δ et Θ ont été fixés à 5 et 3 respectivement. Cette configuration adoptée a permis d'avoir 15 sous-ensembles d'apprentissages de 25% de la base d'exemples, semblable à [Bradford *et al.*1998], et 3 échantillons de tests avec 50% de la base d'exemples. La figure 5.2 illustre le processus d'échantillonnage que nous avons appliqué.

5.2.2 L'encodage

Comme il a été déjà mentionné, $\mathcal{IUDT}$ explore un espace de recherche composé des différents couples qui réunissent les sous-ensembles d'attributs aux sous-ensembles d'exemples d'apprentissages. Formellement, soit $\mathcal{S}_{\mathcal{A}}$ et $\mathcal{S}_{\mathcal{IT}}$ les ensembles de sous-ensembles d'attributs et ceux d'exemples d'apprentissages respectivement. L'espace de recherche $\mathcal{L}$ est l'ensemble de couples issus du produit cartésien $\mathcal{S}_{\mathcal{A}} \times \mathcal{S}_{\mathcal{IT}}$. Cet espace de recherche $\mathcal{L}$ est un ensemble partiellement ordonné, pour chaque couple d'éléments $X(A_i, T_j)$, $Y(A_v, T_w)$ de $\mathcal{L}$: $X \preceq Y$ ssi $A_i \subseteq A_v$ et $T_j = T_w$. En respectant, le prétraitement proposé sur la sous-section 5.2.1, ainsi que cette dernière configuration de l'espace de recherche $\mathcal{L}$, le cardinal de $\mathcal{L}$ est $(\Delta * \Theta)$ * 2^m, pour m le nombre d'attributs de la base de données $\mathcal{D}$. Autrement dit, soit $\mathcal{A}$ l'ensemble des attributs, l'ensemble des couples $\mathcal{L}$ peut être défini comme suit :

$$\mathcal{L} = \{X(A_i, T_j) |\ A_i \subseteq \mathcal{A}\ et\ T_j \in \mathcal{S}_{\mathcal{IT}}\} \tag{5.1}$$

Cependant, par rapport aux valeurs que nous avons sélectionnées $\Delta = 5$ et $\Theta = 3$ pour les expérimentations que nous allons présenté par la suite, le cardinal de $\mathcal{L}$ est $15 * 2^m$.

Example 5.2.1 *Soit une base de données avec deux attributs $\{a, b\}$, et une configuration de prétraitement $\Delta = 5$ et $\Theta = 3$ l'espace de recherche $\mathcal{L}$ illustré sur la figure 5.3, est représenté comme suit :*

$$\mathcal{L} = \{(\{a\}, 1), ..., (\{a\}, 15), (\{b\}, 1), ..., (\{b\}, 15), (\{a, b\}, 1), ..., (\{a, b\}, 15)\}.$$

Pour plus de simplification nous allons décrire un couple comme étant un seul ensemble : $(\{a, b\}, 15) = (a, b, 15)$.

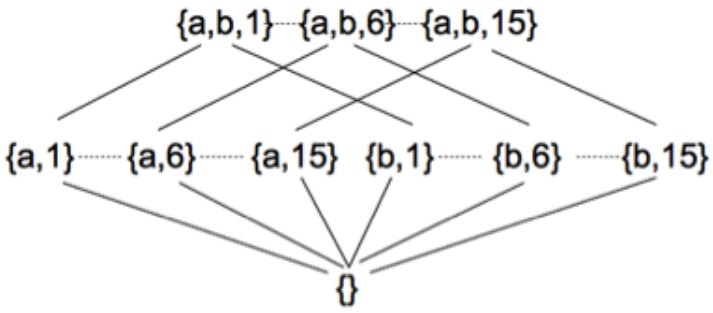

FIGURE 5.3 – Exemple d'un espace de recherche $\mathcal{L}$.

Par conséquent, le problème de découverte du meilleur arbre de décision t^* peut être défini comme suit : soit une base de données $\mathcal{D}$, un langage $\mathcal{L}$ (c.-à-d. espace de recherche),

une fonction objective F, et un prédicat $\mathcal{P}$ qui est satisfait si est seulement si le couple $X(A_i, T_j)$ qui construit l'arbre t^* qui permet de maximiser la fonction F sera atteint. $t^* = \{t((A_i, T_j))|(A_i, T_j) \in \mathcal{L}\ et\ max\ F(t)\}$. La fonction objective F évalue chaque arbre de décision t construit à base du couple (A_i, T_j), avec les sous-ensembles d'apprentissages autres que A_i c.-à-d. $\forall A_x \in \mathcal{S}_{\mathcal{IT}} \backslash A_i$ et l'échantillon du test $b - test$ correspondant au sous-ensemble d'apprentissage A_i. La sélection de l'échantillon de test en respectant la fonction $w : T_j \rightarrow [1, .., \Delta]$, avec $w(T_j) = b$. Par rapport à la configuration, nous proposons la fonction de sélection de $b - test$ est défini comme suit :

$$w(c) = \begin{cases} 1 & if\ \ 1 \leq c \leq 3 \\ 2 & if\ \ 4 \leq c \leq 6 \\ 3 & if\ \ 7 \leq c \leq 9 \\ 4 & if\ 10 \leq c \leq 12 \\ 5 & if\ 13 \leq c \leq 15 \end{cases} \tag{5.2}$$

5.2.3 L'exploration de l'espace de recherche

Pour atteindre l'arbre de décision robuste t^*, $\mathcal{IUDT}$ adopte une exploration en largeur à l'apriori de l'espace de recherche $\mathcal{L}$. Ce processus d'exploration se compose principalement de deux étapes clés : 1) la génération des candidats par jonction et 2) le calcul de la précision de classification pour chaque arbre de décision construit t afin de trouver le meilleur entre eux t^*. En commençant avec le vide $\emptyset$, le premier niveau intermédiaire des candidats L_1. Les candidats de L_1 sont tous les couples (e, i) composés d'un attribut e et un indicateur d'un sous-ensemble d'apprentissage i. Pour chacun des candidats de L_1 la fonction objective F est calculée. Par la suite, en respectant les résultats des candidats de L_1, que ceux satisfaisant les prédicats IRP (Incremental Relevance Property) et PRP (Proportional Relevance Property) sont gardés pour former le premier niveau C_1. À partir du niveau C_1 le niveau intermédiaire L_2 est généré par la jonction de chaque deux candidats qui diffèrent par un seul attribut. Encore, par le raffinement de L_2 le deuxième niveau est généré. À partir des candidats d'un niveau C_k, les candidats du niveau intermédiaire L_{k+1} vont être raffinés pour former le niveau L_{k+1}. Enfin, ce processus est répété jusqu'à ce qu'il n'existe aucun nouveau candidat au niveau C_{k+1} (c.-à-d. $C_{k+1} = \emptyset$).

$$Accuracy(t(X, i)) = Moy\Big(\sum_{a \neq i}^{I} Accuracy(t(X, i), a) + Accuracy(t(X, i), w(i))\Big). \tag{5.3}$$

Pour chaque couple (X, i) exploré par $\mathcal{IUDT}$, un arbre de décision t est construit en utilisant uniquement le sous-ensemble d'apprentissage i et le sous-ensemble d'attributs X. Comme il a été déjà mentionné, la construction de l'arbre implique l'application d'un ordre en sélectionnant les positions d'attributs (p. ex. Gini index, impurity-based criteria,

twoing criterion). Par rapport à ce stade, $\mathcal{IUDT}$ permet d'avoir deux possibilités : soit l'utilisation d'une construction personnalisée où l'utilisation d'un modèle prédéfini (p. ex. BFTree, J48, REPTree, SimpleCart), la seule contrainte est de désactiver la phase d'élagage. Chaque arbre de décision construit est évalué en appliquant la fonction objective F définie sur la formule 5.3. Le pseudocode de $\mathcal{IUDT}$ est illustré sur l'algorithme 12.

Input: Une base de données $\mathcal{D}$, les indices Δ, Θ, et les deux prédicats IRP et PRP.
Output: Un arbre de décision optimal $t^*(X, i)$
Prétraitement($\mathcal{D}$, Δ, Θ, $\mathcal{S}_\mathcal{A}$, $\mathcal{S}_{\mathcal{IT}}$, $\mathcal{S}_\mathcal{S}$);
$L_1 = \{(e, i) | e \in \mathcal{S}_\mathcal{A}, i \in \mathcal{S}_{\mathcal{IT}}\}$;
$C_1 = \{(e, i) | (e, i) \in L_1, IRP(t(e, i))\ and\ PRP(t(e, i))\}$;
$t^* = meilleur - arbre(C_1)$;
$i = 1$;
while C_i *non vide* **do**
  $L_{i+1} = \{(Y, i) | \forall (X, i) \in C_i, \forall e \in \mathcal{S}_\mathcal{A}, Y = X \cup e\}$;
  $C_{i+1} = \{(X, i) |\ (X, i) \in L_{i+1}, IRP(t(X, i))\ and\ PRP(t(X, i))\}$;
  $t^{'} = meilleur - arbre(C_{i+1})$;
  if $Accuracy(t^*) < Accuracy(t^{'})$ **then**
    $t^* = t^{'}$;
  end
  $i = i + 1$;
end
return t^*;

Algorithm 12: | *La recherche du meilleur arbre decision non* élagué

Jusqu'à ce jour, le principal problème d'un processus de découverte de motifs intéressants est principalement la richesse du langage (c.-à-d. le nombre de motifs au sien d'un espace de recherche). Cette richesse rend la phase d'exploration très coûteuse et en pratique même impossible. L'espace de recherche exploré par $\mathcal{IUDT}$ ne fait pas exception à la règle due au nombre énorme de couples qui dépend fortement du nombre d'attributs. Pour faire face à ce problème, pour le cas de la sélection d'attributs la littérature propose soit de minimiser le nombre de motifs à évaluer [Gunopulos *et al.*1997], ou d'appliquer une exploration randomisée de l'espace de recherche . Pour faire face à cette difficulté, $\mathcal{IUDT}$ propose de garder pour chaque niveau que les couples satisfaisant les deux conditions de sélections représentées par les prédicats de sélections IRP et PRP. Pour la première condition, IRP n'est satisfait que si le candidat testé (c.-à-d. la performance de l'arbre de décision) est strictement meilleur que celles des deux couples qui ont été joints à sa génération. Alors que pour la seconde condition, elle consiste à sélectionner que les candidats

qui sont à un seuil Υ du meilleur candidat au même niveau. Pour les expérimentations réalisées un taux de 5% a été fixé pour cette seconde condition (c.-à-d. PRP).

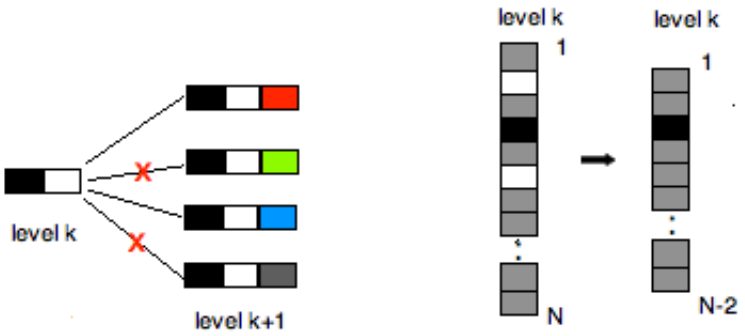

FIGURE 5.4 – Des exemples de : IRP, PRP.

La figure 5.4 illustre un exemple du comportement des deux prédicats IRP et PRP. Soit un candidat d'un niveau k et un groupe de ses spécialisations (fils) au niveau k+1, en appliquant la sélection par IRP pour ses fils, juste ceux qui prouvent une croissance de performance (c.-à-d. F) sont gardés. À gauche de la figure, les candidats issus de l'ajout d'attributs vert et gris sont éliminés. À partir des candidats sélectionnés par IRP le candidat le plus robuste est marqué. Ensuite, tous les candidats ayant une performance strictement inférieure à celle du robuste moins le taux Υ. À droite de la figure en question, le candidat robuste est celui de couleur noire et les candidats en gris satisfont PRP et ceux en blanc ne le satisfont pas.

5.3 Résultats expérimentaux

Dans cette section, les résultats expérimentaux de l'approche $\mathcal{IUDT}$ sont présentés. $\mathcal{IUDT}$ a été implémentée sous Java en utilisant les plateformes WEKA [Hall *et al.*2009] et GUAVA Google library [Bourrillion and others2010]. Afin de tester l'efficacité de $\mathcal{IUDT}$, un ensemble de comparaisons a été conduit. Ces comparaisons ont été réalisées sur 10 bases de la collection UCI [Blake and Merz1998]. Les résultats de $\mathcal{IUDT}$ sont comparés avec ceux : 1) des implémentations originaires des arbres de décisions sous WEKA c.-à-d. des arbres de décisions construits en considérant une phase d'élagage. Nous allons les noter DTP (Decision Tree construction technique with use of Pruning phase). 2) de notre implémentation d'un algorithme qui n'utilise que la sélection d'attributs à la warpper pour l'amélioration de la construction d'un arbre de décision non élagué. Cet

algorithme est appelé IUDTAS (Improved Unpruned Decision Tree construction using only Attribute Selection). 3) de notre implémentation d'un second algorithme qui ne considère que l'échantillonnage de la base de données pour l'amélioration de la construction d'un arbre de décision non élagué. Cet algorithme est appelé IUDTSD (Improved Unpruned Decision Tree construction using only Sampling Data).

TABLE 5.1 – Les caractéristiques des bases de données utilisées pour les expérimentations.

Les bases de données	*No Classes*	*No Attributs*	*No Exemples*	*% erreur de base*
Zoo	7	17	101	59.40
Glass	7	9	214	35.52
Sonar	2	60	208	46.64
Ecoli	8	7	336	57.44
Diabetes	2	8	768	34.90
Hepatitis	2	19	155	20.65
Tic-tac-toe	2	9	958	34 ;65
Breast-cancer	2	9	286	29.72
Primary-tumor	21	17	339	75.22
Waveform-5000	3	40	5000	66.16

Les bases de données utilisées sont présentées au niveau de la Table 5.1. La colonne $\%erreurdebase$ montre le pourcentage d'erreur obtenu, si la classe la plus importante est prédite à tous les coups. Les bases de données sélectionnées représentent différents domaines d'applications. Cependant, la Table 5.2 présente trois algorithmes de construction d'arbres de décisions appelés J48, SimpleCart, et REPTree. Ces trois arbres de décisions ont été choisis vu les différentes techniques d'élagage qui seront adoptées.

Comme nous l'avons déjà mentionné, $\mathcal{IUDT}$ permet de traiter des modèles d'arbres de décisions déjà existants. Les expérimentations présentées dans cette thèse sont menées en utilisant les modèles sur la Table 5.2. Ces arbres de décisions sont implémentés dans leurs paramètres standards en suspendant la phase d'élagage (DTs non élagué).

Les Tables 5.3, 5.4 et 5.5 listent la précision de classification pour les modèles J48, REPTree, et SimpleCart, respectivement. Ces résultats sont issus de l'application d'$\mathcal{IUDT}$ aux 10 bases de données sur la Table 5.1. Toutes les tables illustrent que généralement la totalité des attributs de la base de données n'est jamais utilisée lors de la construction d'arbres de décisions, et seulement un sixième des attributs est utilisé pour les cas de bases

TABLE 5.2 – Les caractéristiques standards des arbres de décisions sous WEKA.

DTs	Critère de séparation	Méthodes d'élagages	Les paramètres par défauts
J48 (C 4.5)	Gain ratio	Error-Based	-La confidence d'élagage est à 0.25 ; -Deux exemples au minimum pour chaque feuille ; -Un seul échantillon est utilisé pour l'élagage ;
REPTree	Gain ratio	Reduced-Error	-Deux exemples au minimum pour chaque feuille ; -Un seul échantillon est utilisé pour l'élagage.
SimpleCart	Gini index	Error-Complexity	-Division binaire pour les attributs de type nominal ; -Deux exemples au minimum pour chaque feuille ; -Un seul échantillon est utilisé pour l'élagage ; -Cinq échantillons sont utilisés pour une validation croisée.

TABLE 5.3 – Les résultats des J48 issues d'$\mathcal{IUDT}$

Les bases de données	# Les attributs	La taille	La précision
Zoo	3	11	90.50
Glass	5	29	70.56
Sonar	7	17	91.10
Ecoli	5	15	85.26
Diabetes	5	25	80.92
Hepatitis	6	15	84.09
Tic-tac-toe	8	73	84.70
Breast-cancer	5	51	79.02
Primary-tumor	10	42	53.40
Waveform-5000	7	179	83.80

de données ayant un nombre d'attributs de plus en plus large. Par ailleurs, les résultats en précision de classification illustrent que globalement les performances des arbres de décisions surclassent les % *erreurs de base*.

La Table 5.6 illustre la précision de classification de l'ensemble d'arbres de décisions en utilisant les paramètres par défaut (**DTP**). Ces arbres sont construits en respectant le processus complet en poussant les arbres, et les élagués par la suite. La moitié (c.-à-d. 50%) d'exemples des bases de données a été utilisée pour la phase de construction et l'autre

TABLE 5.4 – Les résultats des REPTree issues d'$\mathcal{IUDT}$

Les bases de données	# Les attributs	La taille	La précision
Zoo	5	13	95.00
Glass	5	27	76.40
Sonar	6	17	91.58
Ecoli	5	15	83.33
Diabetes	6	57	77.60
Hepatitis	3	9	79.22
Tic-tac-toe	9	76	81.83
Breast-cancer	5	68	78.49
Primary-tumor	10	40	56.30
Waveform-5000	6	295	83.08

TABLE 5.5 – Les résultats des SimpleCart issues d'$\mathcal{IUDT}$

Les bases de données	# Les attributs	La taille	La précision
Zoo	5	13	95.00
Glass	4	19	72.66
Sonar	6	17	94.95
Ecoli	3	19	85.56
Diabetes	5	61	78.71
Hepatitis	6	17	89.93
Tic-tac-toe	9	49	93.05
Breast-cancer	6	31	77.92
Primary-tumor	10	43	52.51
Waveform-5000	6	231	82.19

moitié est utilisée pour tester les performances des modèles construits. Afin d'avoir des comparaisons représentatives par rapport à $\mathcal{IUDT}$, chacune des bases de données a été échantillonnée aléatoirement Δ fois (c.-à-d. comme Δ a été fixé par 5 pour $\mathcal{IUDT}$, nous optons pour 5 échantillons d'apprentissages et 5 autres de tests de 50%). Les moyennes des précisions de classifications et des tailles des modèles construits par rapport aux cinq échantillons de tests $i - test$ sont illustrées sur la Table 5.6. Les résultats varient d'un modèle à un autre pour cette expérience. Pour la taille, nous remarquons que les J48 sont plus larges que les REPTree et les SimpleCart. Ces résultats sont expliqués naturellement par les méthodes d'élagages utilisés. Contrairement à Reduced-error pruning (REPTree) et Error Complexity Pruning (SimpleCart), la technique EBP adoptée par J48, opte la

possibilité de greffer l'un des sous-arbres d'un sous-arbre qui a été sélectionné pour être élagué. Par conséquent, la précision de classification des modèles J48 est meilleure pour six bases de données comparativement aux modèles REPTree et SimpleCart.

TABLE 5.6 – Les Results de **DTP**

Les données	J48 Taille	J48 Précision	REPTree Taille	REPTree Précision	SimpleCart Taille	SimpleCart Précision
Zoo	13	95.20	1	43.60	1	43.60
Glass	35	80.56	9	72.33	9	69.90
Sonar	19	89.42	3	77.88	9	80.76
Ecoli	25	82.26	13	82.26	15	82.02
Diabetes	31	81.40	39	80.57	5	77.55
Hepatitis	9	87.01	7	85.71	17	90.12
Tic-tac-toe	97	84.88	64	79.16	45	92.94
Breast-cancer	20	71.04	1	72.16	1	72.16
Primary-tumor	46	52.30	20	44.85	21	48.04
Waveform-5000	341	85.36	87	80.28	49	79.36

De plus, ces résultats nous présentent une idée globale sur les problèmes de sur et sous élagage que nous pouvons rencontrer avec certaines bases de données. Pour le cas d'un sur élagage, les résultats des modèles des REPTree et SimpleCart par rapport aux bases de données Zoo et Breast-cancer présentent un exemple parfait. Par ailleurs nous pouvons distinguer les résultats des modèles J48 et REPTree pour la base de données Ecoli, les précisions de classifications sont équivalentes, mais le modèle J48 et d'une plus grande taille par rapport au modèle REPTree. Nous pouvons donc conclure que le modèle de J48 est un cas d'un sous élagage.

Dans le but de démontrer l'efficacité de l'approche proposée, $\mathcal{IUDT}$ est comparée à un algorithme **IUDTSD** qui n'utilise que l'échantillonnage des données pour l'amélioration de la performance des arbres construits. Soit un modèle de construction d'arbres de décisions X c.-à-d. J48, ou REPTree, ou SimpleCart, **IUDTSD** construit avec chacun des sous-exemples d'apprentissages justes en poussant les arbres sans les élaguer. Ensuite, chacun des arbres est évalué sur l'ensemble des échantillons de tests pour enfin récupérer la moyenne. Les résultats d'expérimentations d'**IUDTSD** sont présentés sur la Table 5.7. Les résultats illustrent que les arbres de décisions sont plus larges que ceux issus de **DTP**, mais les performances sont en faveur de ceux issus de **IUDTSD**.

TABLE 5.7 – Les résultats d'**IUDTSD**

Les données	J48 Taille	J48 Précision	REPTree Taille	REPTree Précision	SimpleCart Taille	SimpleCart Précision
Zoo	11	94.00	1	45.00	7	55.50
Glass	17	74.29	25	74.06	17	71.72
Sonar	11	85.09	9	84.61	9	83.89
Ecoli	17	84.11	9	84.22	29	84.52
Diabetes	27	81.70	45	81.90	47	82.29
Hepatitis	11	89.61	9	87.66	13	88.61
Tic-tac-toe	73	84.70	91	83.97	49	92.95
Breast-cancer	27	77.62	61	73.07	7	77.27
Primary-tumor	49	50.88	50	50.14	45	54.28
Waveform-5000	197	83.98	191	84.47	155	82.48

Les performances d'un algorithme qui n'applique que la sélection d'attributs sans faire appel à l'échantillonnage des données **IUDTAS** sont présentées sur les Tables 5.8-5.10. **IUDTAS** est un algorithme wrapper qui traverse l'espace de recherche composé des sous-ensembles d'attributs. Pour un sous-ensemble d'exemples d'apprentissages de %50, au niveau de chacun des sous-ensembles explorés A un arbre de décision t est construit en utilisant que les attributs A. L'arbre t est évalué sur le même ensemble des échantillons de tests utilisés pour $\mathcal{IUDT}$, DTP, et **IUDTSD**.

TABLE 5.8 – Les résultats des J48 issus d'**IUDTAD**

Les bases de données	# Les attributs	La taille	La précision
Zoo	5	11	93.60
Glass	2	27	79.81
Sonar	3	13	87.30
Ecoli	3	17	80.71
Diabetes	3	11	78.95
Hepatitis	3	9	85.45
Tic-tac-toe	5	124	84.63
Breast-cancer	2	12	75.94
Primary-tumor	8	38	53.72
Waveform-5000	6	189	81.21

Les Tables 5.11–5.13 présentent les comparaisons des résultats issus d'$\mathcal{IUDT}$ par rapport aux **DTP** (Table 5.11), **IUDTSD** (Table 5.12), et **IUDTAS** (Table 5.13). Le symbole "+" signifie que l'approche proposée a produit de meilleures performances que les autres méthodes c.-à-d. **DTP**, **IUDTSD**, et **IUDTAS**. D'autre part, le symbole "-" est utilisé. Nous notons que les cas des sur élagages où les arbres de décisions de taille une (c.-à-d. Zoo et Breast-cancer) sont considérés non performants, ainsi que deux précisions de classifications sont considérés équivalente si leur différence appartient à l'intervalle $[-1, +1]$ %. Autrement, l'arbre de décision avec la meilleure performance est sélectionné comme le meilleur.

Les comparaisons sur Table 5.11 indiquent clairement que l'approche proposée $\mathcal{IUDT}$ est la plus performante par rapport à **DTP** (les cas de REPTree et SimpleCart). Alors que dans le cas de J48, chacune des deux approches est plus performante pour quatre bases de données. Les arbres de décisions issus de l'approche **DTP** sont plus petit pour les modèles REPTree et SimpleCart, mais pour le J48, les arbres de décisions de $\mathcal{IUDT}$ sont plus petits.

La Table 5.12 compare les résultats issus d'$\mathcal{IUDT}$ avec ceux issus d'**IUDTSD**. Les précisions de classifications des arbres issues d'$\mathcal{IUDT}$ sont largement meilleures que ceux issus d'**IUDTSD**, mais contrairement aux résultats des précisions, les tailles des arbres illustrent que l'utilisation de l'échantillonnage (c.-à-d. **IUDTSD**) est plus avantageux. La table indique aussi que généralement les résultats de précisions d'$\mathcal{IUDT}$ sont meilleurs par rapport aux REPTree et SimpleCart. Ces arbres sont plus larges que ceux issus

TABLE 5.9 – Les résultats des REPTree issus d'**IUDTAS**

Les bases de données	# Les attributs	La taille	La précision
Zoo	4	11	91.20
Glass	2	45	80.93
Sonar	3	27	87.11
Ecoli	3	53	84.52
Diabetes	3	89	84.79
Hepatitis	3	27	89.09
Tic-tac-toe	4	94	78.91
Breast-cancer	2	21	75.94
Primary-tumor	8	54	53.60
Waveform-5000	8	505	86.01

TABLE 5.10 – Les résultats des SimpleCart issus d'**IUDTAS**.

Les bases de données	# Les attributs	La taille	La précision
Zoo	5	11	94.40
Glass	2	47	82.80
Sonar	4	27	91.73
Ecoli	3	45	81.66
Diabetes	2	145	82.60
Hepatitis	3	27	89.09
Tic-tac-toe	6	105	90.43
Breast-cancer	2	17	75.94
Primary-tumor	7	61	55.14
Waveform-5000	8	233	86.41

TABLE 5.11 – Les résultats de comparaisons de **DTP** et $\mathcal{IUDT}$.

Les données	J48		REPTree		SimpleCart	
	Taille	Précision	Taille	Précision	Taille	Précision
Zoo	+	-	+	+	+	+
Glass	+	-	-	+	-	+
Sonar	+	+	-	+	-	+
Ecoli	+	+	-	+	-	+
Diabetes	+	=	-	-	-	+
Hepatitis	-	-	-	-	=	=
Tic-tac-toe	+	=	-	+	-	=
Breast-cancer	-	+	+	+	-	+
Primary-tumor	+	+	-	+	-	+
Waveform-5000	+	-	-	+	-	+

d'**IUDTSD** . Pour le cas de J48, les comparaisons sont équitables soit pour les précisions ou par rapport aux tailles. Enfin, la table démontre que l'utilisation que de l'échantillonnage améliore le processus de réduction de la taille (c.-à-d. complexité des arbres), mais cela est au coût de la performance (c.-à-d. précision).

La Table 5.13 compare les résultats des arbres de décisions issus d'$\mathcal{IUDT}$ avec ceux issus d'**IUDTAS**. Clairement, cette comparaison démontre un surclassement d'$\mathcal{IUDT}$.

TABLE 5.12 – Les résultats de comparaisons de **IUDTSD** et $\mathcal{IUDT}$.

Les données	J48		REPTree		SimpleCart	
	Taille	Précision	Taille	Précision	Taille	Précision
Zoo	=	-	+	+	-	+
Glass	-	-	-	+	-	=
Sonar	-	+	-	+	-	+
Ecoli	+	+	-	=	+	+
Diabetes	+	=	-	-	-	-
Hepatitis	-	-	=	-	-	+
Tic-tac-toe	=	=	+	-	=	=
Breast-cancer	-	+	-	+	-	+
Primary-tumor	+	+	+	+	+	-
Waveform-5000	+	=	-	+	-	=

TABLE 5.13 – Les résultats de comparaisons de **IUDTAS** et $\mathcal{IUDT}$.

Les données	J48		REPTree		SimpleCart	
	Taille	Précision	Taille	Précision	Taille	Précision
Zoo	-	=	-	+	-	+
Glass	-	-	+	-	+	-
Sonar	-	+	+	+	+	+
Ecoli	+	+	+	-	+	+
Diabetes	-	+	+	-	+	-
Hepatitis	-	-	+	-	+	=
Tic-tac-toe	+	=	+	+	+	+
Breast-cancer	-	+	-	+	-	+
Primary-tumor	-	-	+	+	+	-
Waveform-5000	+	+	+	-	+	-

Généralement, la table illustre que $\mathcal{IUDT}$ apporte de meilleurs résultats soit par rapport aux précisions et les tailles des arbres pour les modèles REPTree et SimpleCart. À l'exception, le cas des modèles J48, les tailles des arbres issues d'**IUDTAS** sont plus petits. En conclusion, ces résultats prouvent que l'utilisation de la combinaison de la sélection d'attributs et l'échantillonnage de la base de données permet la construction d'arbres de décisions à la fois robustes et de petites tailles.

Conclusion du chapitre

La popularité des arbres de décisions provient de leurs simplicités, leurs facilités, et aussi leurs ressemblances au raisonnement humain. Cependant, chacun des modèles proposés a des avantages, des limites, ainsi des particularités, ce qui rend juste le fait de choisir entre eux difficile à justifier. Principalement, un utilisateur doit bien connaitre ses données ainsi que leurs qualités. Bien qu'il doit étudier beaucoup de techniques utilisés par ces différents modèles pour les phases d'augmentation et d'élagage, pour pouvoir enfin faire un choix justifié avec de grandes chances d'être le bon modèle.

Lors de la construction des arbres de décisions, la phase d'élagage est très utile afin de réduire la complexité des arbres de décisions et de contourner le problème de sur apprentissage. Mais malheureusement, ces méthodes affectent négativement la performance des arbres surtout pour les cas des bases de données de petite taille. Dans ce chapitre nous avons présenté une approche qui permet de surpasser les problèmes pour lesquels l'élagage a été proposé. Cette approche proposée a permis d'améliorer la qualité de l'arbre lors de la phase d'augmentation afin qu'il n'ait pas besoin d'être élagué. Cela a été réalisé en cherchant la combinaison optimale composée d'un sous-ensemble d'attributs et un autre d'exemples d'apprentissage qui nous permet de construire un arbre de décision sans avoir recours à une phase d'élagage.

Conclusion générale

Comme les algorithmes combinatoires sont au coeur de nombreux domaines d'applications, tels que l'apprentissage, le data mining, les bases de données, la représentation des connaissances et le raisonnement, etc. L'objectif de cette thèse était principalement d'étudier la découverte des sous-graphes connexes fréquents. Dans ce cadre, nos contributions ont porté sur la proposition de deux approches pour la découverte des sous-graphes connexes fréquents à étiquetage d'arêtes non redondant. L'apport des deux techniques proposées a été concrétisé suivant deux niveaux. Premièrement, la restriction de l'espace de recherche. Deuxièmement, la réduction de la complexité des tests d'isomorphisme de graphes et de sous-graphes. Autres que ces apports, les techniques adoptées pour la représentation des graphes ont permis d'illustrer l'impact de la représentation des données vis-à-vis la résolution des problèmes d'isomorphisme de graphes et de sous-graphes d'une part, et celle de la génération des nouveaux candidats connexes d'autre part. Partiellement, nous nous sommes intéressés aussi à la construction d'un arbre de décision sous contraintes comme étant une spécialisation de la découverte des modèles intéressants. Nous avons proposé un algorithme de découverte d'une combinaison de sous-ensembles d'exemples d'apprentissage et d'attributs optimale (intéressante) qui permet la construction du meilleur arbre de décision en respectant certaines contraintes.

Contributions

Découverte de sous-graphes connexes fréquents

Les deux approches proposées pour la découverte de sous-graphes connexes à étiquetage d'arêtes non redondant **ECERCSgl**, et **ADARCSgl**, [Karabadji and Seridi2014b], [Karabadji and Seridi2014a] ont chacune leurs avantages et leurs limites. Par rapport à l'accessibilité forte de l'espace de recherche, les deux approches partagent le même principe vis-à-vis l'exploitation de cette dernière propriété pour la restriction de l'espace de recherche. Cette restriction consiste à réduire le nombre des sous-graphes non intéressants (c.-à-d. les fréquents non fermés) à visiter. En pratique cette réduction est possible du fait que pour chacun des sous-graphes fermés, il peut être atteint en escaladant l'espace

de recherche en ajoutant une arête à la fois à partir d'un sous-graphe fermé de ses prédécesseurs. Ce principe permet de tenter d'atteindre chacun des sous-graphes fermés en sautant directement à partir d'un autre fermé de ses prédécesseurs en respectant la fermeture descendante de l'espace de recherche vis-à-vis de la fréquence.

Cependant, ces approches proposées adoptent deux représentations mémoires des graphes distincts. Ces deux représentations sont : une représentation ensembliste pour **ECERCSgl**, et représentation par tableaux d'adjacences pour **ADARCSgl**. Ces deux choix ont été motivés principalement : a) par la facilité d'encodage des représentations ensemblistes comme étant des ensembles d'items. Ce qui permet de tester l'isomorphisme de graphes et de sous-graphes en temps polynomial. Ainsi, le calcul de l'opération de fermeture se résume en l'intersection d'un ensemble d'ensembles d'items. b) par la représentation des graphes comme des tableaux d'adjacences, cela permet de résoudre efficacement le test d'isomorphisme de sous-graphes dans le moyen des cas, et même en un temps linéaire pour le meilleur des cas. En contrepartie, le coût de stockage en mémoire pour la représentation ensembliste, ainsi que les coûts des opérations d'encodages et de décodages, des graphes et des sous-graphes sont corrélativement liés aux tailles et aux densités des sous-graphes manipulés. Alors que la représentation par tableaux d'adjacences complique le calcul direct de l'opération de fermeture, alors à partir de chacun des sous-graphes fermés, un algorithme d'exploration en profondeur d'abord est appliqué pour atteindre ses successeurs fermés.

Découverte de modèles à base de contraintes

Nous avons proposé une approche de découverte d'une combinaison composée d'un sous-ensemble d'attributs et un autre d'exemples d'apprentissages qui permet la construction d'un arbre de décision qui maximise la performance par rapport à un ensemble indépendant d'exemples de test. Cette approche permet la construction de l'arbre de décision en utilisant seulement un sous-ensemble d'exemples représentatif. Cet échantillonnage permet d'éviter les contres exemples, les ignorances, etc. De même pour la sélection d'attributs, cela nous permet de limiter les choix de sélection d'attributs de division, ainsi que l'élimination des attributs pour lesquels les valeurs sont problématiques.

Perspectives

Dans la continuité directe de notre travail de thèse, nos perspectives se décomposent en deux points, la découverte de sous-graphes intéressants et l'amélioration des performances des modèles de classifications.

Il serait intéressant de regarder encore l'encodage des graphes comme étant des ensembles d'items [Thomas *et al.*2009]. Comme nous l'avons illustré, cet encodage a permis de résoudre un problème majeur des graphes, qui est l'isomorphisme de sous-graphes pour une classe particulière. Cependant, il est très intéressant d'appliquer ce même encodage pour d'autres cas parmi eux le cas général. Mais pour pouvoir encoder un graphe, il faut faire très attention aux symétries soit des arêtes des couples d'arêtes et des triplets d'arêtes. Le coût de traitement des symétries est très élevé, mais les avantages d'avoir une représentation en un ensemble d'items sont multiples.

Il est aussi intéressant de trouver des applications réelles pour lesquelles les données sont représentées par des graphes à étiquetage d'arêtes non redondant. Comme la détermination des itinéraires les plus populaires dans une ville donnée. Dans ce contexte, notre approche pourra se révéler précieuse pour détecter les itinéraires fréquents fermés.

La contribution majeure de [Fredman and Khachiyan1996] pour générer tous les transversaux d'un hypergraphe pour un temps quasi polynomial, ainsi que la définition de deux classes $\mathcal{WRAS}$ et $\mathcal{EWRAS}$ par [Nourine and Petit2012] qui garantit un délai quasi polynomial d'énumération de deux motifs maximaux successifs. Malheureusement, cette garantie est posée sous la contrainte de trouver un plongement du langage des motifs tels que le cardinal de l'ensemble des images en surplus est : soit polynomial (c.-à-d. le cas $\mathcal{WRAS}$), soit que les images extras sont à l'écart de celles des motifs initiaux, ce qui permet de les distinguer plus facilement (c.-à-d. le cas $\mathcal{EWRAS}$). Comme application, seul le problème de découverte de sous-séquences rigides a été classé en $\mathcal{EWRAS}$. Il est intéressant de définir un encodage qui permet de classer l'un des problèmes de découverte pour qui l'espace de recherche ne correspond pas à un treillis ni à un produit de chaînes.

Enfin, notre méthode de construction d'un arbre de décision optimal s'appuie sur deux contraintes de réduction de l'espace de recherche. De fait, il est possible d'utiliser n'importe quelle contrainte exploitable afin de favoriser la restriction de certaines combinaisons ayant des caractéristiques particulières. En plus même, par rapport à l'étape de prétraitement, l'échantillonnage aléatoire appliqué peut être remplacé par un autre structuré afin d'éliminer certaines données ayant des caractéristiques non désirées.

Annexes

Annexe A

A.1 Diagnostic de pannes en une machine tournante

Le diagnostic de pannes en une machine tournante est un problème célèbre pour lequel beaucoup de travaux se sont intéressés [Kankar *et al.*2011], [Djebala *et al.*2008], [Khelf *et al.*2013]. Dans cette section, nous considérons l'application de $\mathcal{IUDT}$ au problème de diagnostic de pannes en une machine tournante. Pas mal de pannes qui peuvent affecter le bon fonctionnement d'une machine tournante sont produites expérimentalement sur un bon d'essais. Le banc utilisé est installé au niveau du centre de recherche URASM-CSC Annaba. La figure1 illustre les détails du banc. L'objective de cette application est l'extraction d'information à partir des capteurs de vibrations pour prédire l'état actuel d'une machine (la classe).

L'approche proposée $\mathcal{IUDT}$ est utilisée dans le but de construire un ensemble de règles de décisions efficaces non complexes. La figure2 illustre le schématique d'application d'$\mathcal{IUDT}$.

A.2 Description du banc et des essais menés

Le banc schématisé sur la figure 2 est installé au niveau du centre de recherche URASM-CSC Annaba. Le banc est constitué d'un châssis en aluminium, sur lequel trois lignes d'arbres sont montées par l'intermédiaire de six paliers à roulements. Une transmission par engrenage et utilisée pour lier l'arbre *A1* à l'arbre *A2*, par l'intermédiaire de deux roues dentées (60 et 48 dents), et une transmission par courroie pour le même effet entre l'arbre *A2* et l'arbre *A3*. Trois disques sont installés sur les arbres *A1* et *A3* permettant de simuler des défauts de balourd.

Le système est accouplé à un moteur à induction d'une puissance de 0,18 kW, et d'une vitesse de rotation nominale de 1500 tr/min, contrôlé par un variateur de vitesse.

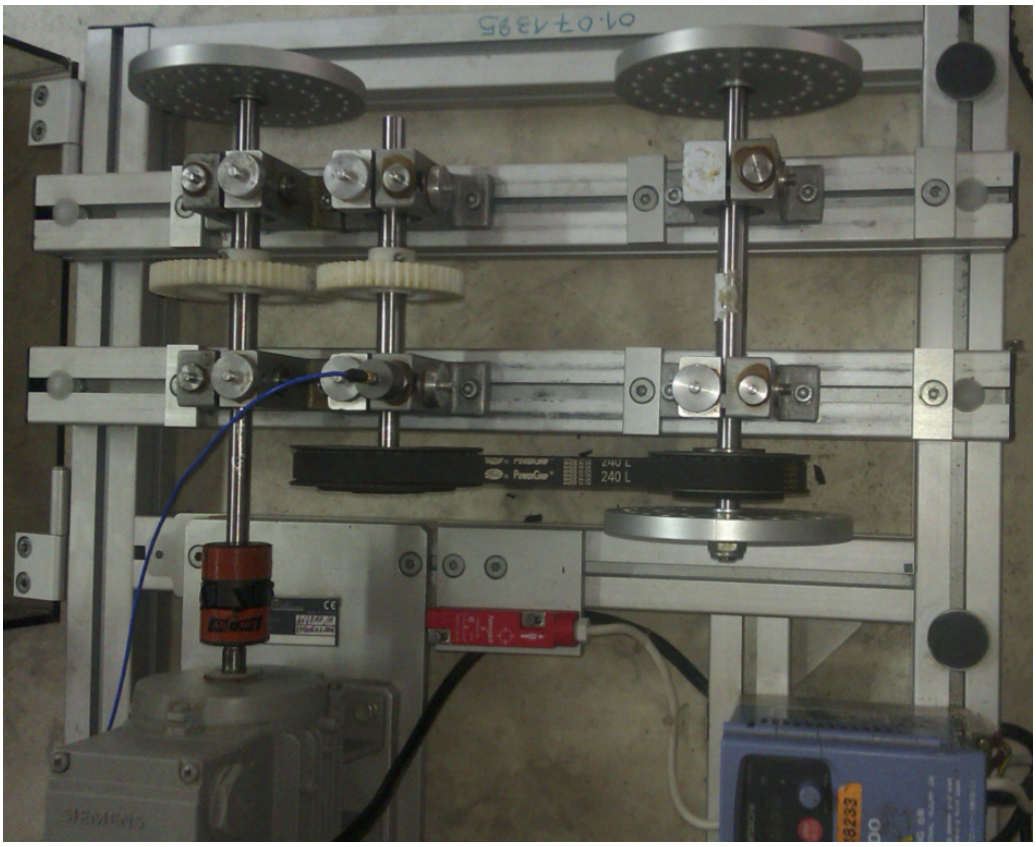

FIGURE A.1 – Schématisation du banc d'essais URASM.

Quatre conditions de fonctionnement ont été expérimentées : sans la présence de défauts ; avec la présence d'un défaut de balourd matérialisé par un poids additionnel sur le disque 1 ; un défaut d'engrenage produit par un remplissage entre deux dents ; et un dernier défaut simulé par l'arrachage de six dents de la courroie crantée. Pour chaque condition de fonctionnement, trois vitesses de rotation supposées stationnaires, ont été observés, à savoir, 300, 600 et 1200 tr/min.

Des signaux vibratoires ont été extraits via un accéléromètre, vissé sur l'embase collée sur le palier *P1*. Les signaux ont été acquis par la suite via un collecteur multivoie OROS-25, permettant un conditionnement correct des signaux «filtrage anti-repliement, amplification». Les signaux ont été relevés sur des fenêtres temporelles de 400 millisecondes, avec une fréquence d'échantillonnage de 5120 Hz, et ont été transmis par la suite à un ordinateur équipé du logiciel OROS763.

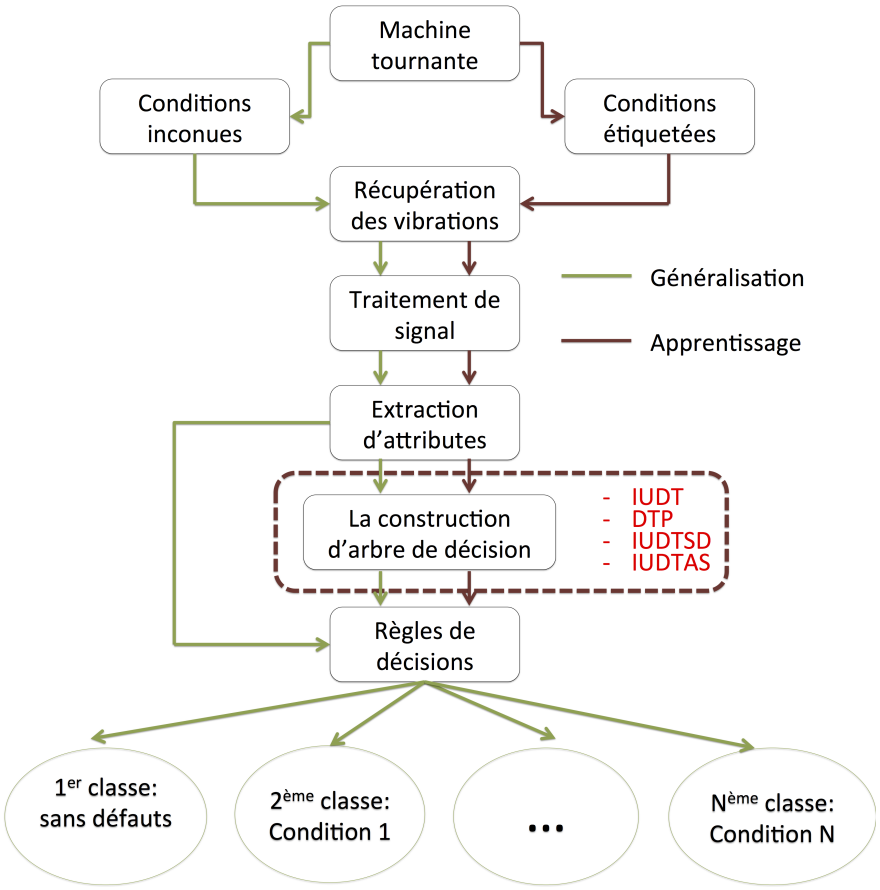

FIGURE A.2 – Optimisation de la construction d'arbres de décisions pour le diagnostic de pannes en une machine tournante.

A.2.1 Ondelettes

Contrairement à la *STFT*, la transformée d'ondelette est une méthode de traitement de signaux possédant une résolution adaptative à la taille de l'objet ou du détail analysé. De même que pour la transformée de Fourier cette technique décompose le signal dans une base de fonctions particulières, sauf que ces fonctions « appelées ondelettes », contrairement aux fonctions sinusoïdales de l'analyse de Fourier, sont des fonctions oscillantes au sens large et peuvent être rapidement amorties. Par ailleurs, les ondelettes possèdent la propriété de pouvoir être bien localisées en temps ou en fréquence, une ondelette dilatée observe les composantes basses fréquences nécessitant une large fenêtre temporelle, alors qu'une ondelette contractée observe les composantes hautes fréquences où une haute résolution temporelle est requise.

Les éléments de base de la transformée en Ondelettes sont des fonctions localisées en temps autour d'un paramètre b et oscillant à une fréquence a. Elles sont générées par

translation b et dilatation a à partir d'une fonction, l'ondelette mère de moyenne nulle , comme montrée dans l'équation A.1.

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right) \tag{A.1}$$

La transformée en Ondelettes d'une fonction $x(t)$ est définie au moyen du produit scalaire montré par l'équation A.2 conduisant à une représentation temps-échelle en fonction des variables décalage et échelle.

$$TOC(a,b) = \frac{1}{\sqrt{a}}\int_{-\infty}^{\infty} X(t) * \psi^*\left(\frac{t-b}{a}\right) dt \tag{A.2}$$

Une discrétisation de la transformée est possible en posant 2^m et $n2^m$ au lieu des paramètres a et b , avec m et n des entiers, comme montré dans l'équation A.3.

$$TOD(m,n) = 2^{\frac{-m}{2}}\int_{-\infty}^{\infty} X(t) * \psi^*\left(2^{-m}(t-n)\right) dt \tag{A.3}$$

Dans le cadre des signaux numériques, l'Analyse Multi-Résolution en Ondelette *(AMRO)* est largement utilisée, permettant de décomposer un signal, en sous-bandes fréquentielles, sans perte ou redondance d'informations Mallat89. Cette technique génère à chaque niveau un coefficient d'approximation contenant l'information (basses fréquences), et un coefficient de détail contenant l'information (hautes fréquences) du signal d'origine. L'opération peut être répétée sur plusieurs niveaux et conduit à la création de l'arborescence montrée dans la figure A.3.

Ces coefficients subissent un sous-échantillonnage lors de l'analyse, pour palier à cela un banc de filtres de synthèse est employé, réalisant les opérations duales de celles effectuées lors de la décomposition et permettant de reconstruire les signaux A Approximation et D Détails satisfaisant la relation suivante :

$$A_{k-1} = A_k + D_k \; et \; X = A_k + \sum_{i \leq k} D_i \quad i \, et \, k \, sont \, des \, entiers \tag{A.4}$$

A.2.2 Extraction d'indicateurs

Á partir de chacune des quatre conditions de fonctionnement, 35 indicateurs ont été extraits. Le facteur crete, le Skweness, le RMS et la variance ont été extraits de la forme temporelle des signaux. Á partir des spectres fréquentiels ont été extraits : l'amplitude du plus grand pic et sa fréquence, l'amplitude du deuxième plus grand pic, l'intervalle

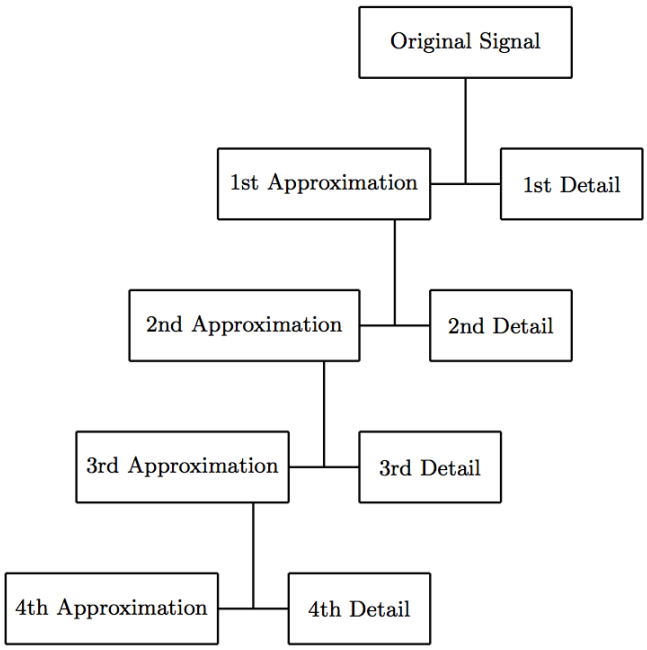

FIGURE A.3 – Arbre de décomposition (AMRO)

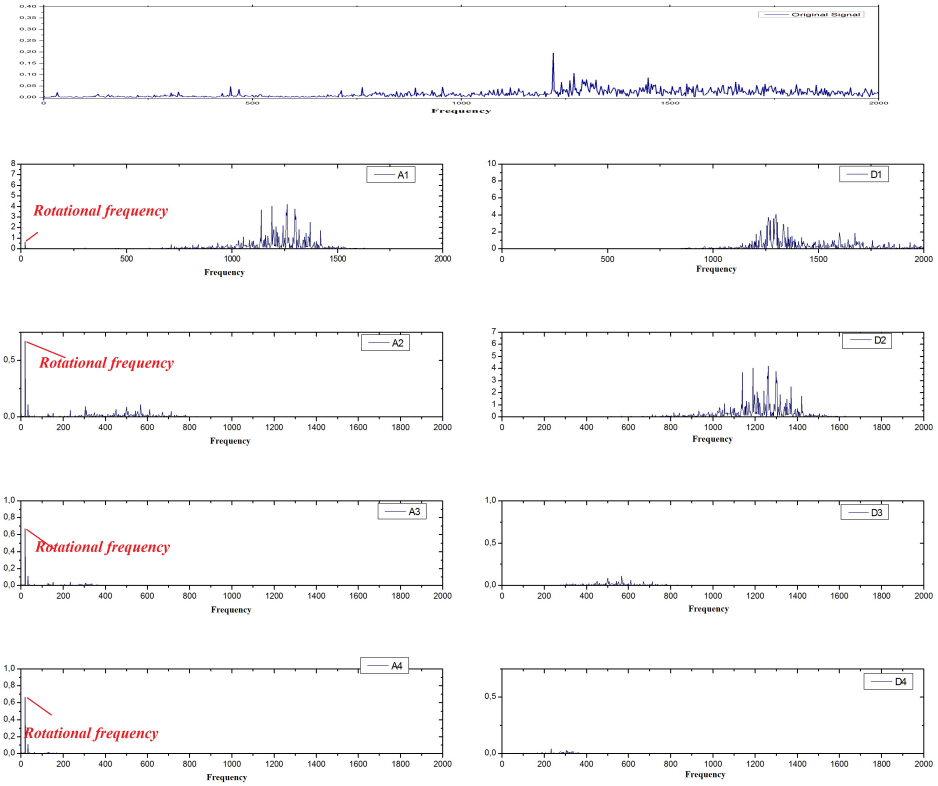

FIGURE A.4 – Les signaux

fréquentiel entre les deux plus grands pics, l'intervalle fréquentiel moyen entre les quatre pics les plus élevés, et le RMS. Ces 11 indicateurs ont aussi été extraits à partir des quatre signaux obtenus après l'application de l'AMRO sur deux niveaux de décomposition, pour un total de 55 indicateurs extraits sur chaque signal originaire. Pour plus de détail sur le processus mené pour l'extraction de signaux et d'indicateurs à partir du banc utilisé pour cette application, consulter la thèse de [KHELF2014].

A.2.3 Les résultats

Les résultats d'expérimentations issues d'$\mathcal{IUDT}$ menés pour les modèles $J48$, $REPTree$ et $SimpleCart$ sont présentés sur la Table 1. Ces résultats confirment la supériorité déjà démontrée en terme de précision de classification. **IUDTSD** s'impose légèrement encore par rapport à la simplicité de ses arbres de décisions, mais cette simplicité est accompagnée à des performances faibles. Par conséquent, **IUDTSD** peut faire cas de sur généralisation.

TABLE A.1 – Diagnostic de pannes en une machine tournante

	IUDT		**DTP**		**IUDTSD**		**IUDTAS**	
	Précession	Taille	Précession	Taille	Précession	Taille	Précession	Taille
J48	89.52	21	91.66	33	92.38	19	96.66	29
REPTree	98.41	21	85.31	27	92.14	19	90.47	29
SimpleCart	95.37	23	92.3	35	90.47	19	96.66	25

Les meilleurs résultats ont été obtenus par le modèle REPTree, les règles de décisions produites à l'issue de ces expérimentations sont présentées au niveau de l'appendice. Les résultats d'$\mathcal{IUDT}$ pour REPTree apportent les meilleures performances par rapport à la précision. La Table 2 liste les indicateurs utilisés pour la construction des arbres de décisions d'$\mathcal{IUDT}$ par ordre d'apparence, ainsi que l'illustration de leurs signaux d'originaires.

TABLE A.2 – Les indicateurs utilisés pour la construction du meilleur arbre de décision

AOT	**Indicateur**	**Le signal originaire**
1	Skewness	Original signal
2	Root Mean Square of the spectra	Original signal
3	Signal Root Mean Square	1st Approximation signal
4	Crest Factor	1st Detail signal
5	Mean Interval between the Frequencies of the Four Maximum Amplitudes	Original signal

Sur la Table 2, nous pouvons illustrer que la majorité des attributs utilisés pour la construction des arbres de décisions sont récupérés du signal originaire et de celui du premier niveau de décomposition. Alors le fait d'exclure les trois dernières phases de décompositions permet d'avoir un significatif gain soit par rapport à la mémoire de stockage, soit en terme d'optimisation du temps de calcul.

Annexe B

B.1 Exemple de construction d'un arbre de décision

Soit l'ensemble d'exemples de la table 1, sur cette sous-section, nous présentons la construction d'un arbre de décision à base du gain d'information comme critère de séparation. L'attribut cible (c.-à-d. la classe) est "Jouer au tennis ? " :

TABLE B.1 – Exemple jouer au tennis ?

Jour	Ciel	Température	Humidité	Vent	Jouer au tennis ?
1	Ensoleillé	Chaude	Élevée	Faible	Non
2	Ensoleillé	Chaude	Élevée	Fort	Non
3	Couvert	Chaude	Élevée	Faible	Oui
4	Pluie	Tiède	Élevée	Faible	Oui
5	Pluie	Fraîche	Normale	Faible	Oui
6	Pluie	Fraîche	Normale	Fort	Non
7	Couvert	Fraîche	Normale	Fort	Non
8	Ensoleillé	Tiède	Élevée	Faible	Non
9	Ensoleillé	Fraîche	Normale	Faible	Oui
10	Pluie	Tiède	Normale	Faible	Oui
11	Ensoleillé	Tiède	Normale	Fort	Oui
12	Couvert	Tiède	Élevée	Fort	Oui
13	Couvert	Chaud	Normale	Faible	Oui
14	Pluie	Tiède	Élevée	Fort	Non

D'abord, une racine doit être sélectionnée. Comme il n'existe pas un attribut qui nous permet de diviser les exemples de la table en deux sous-ensembles purs (c.-à-d. de la même classe), le gain d'information pour chacun des attributs est calculé.

Les résultats du gain d'information sont donnés comme suit :

- Ciel = 0.246
- Humidité = 0.151
- Vent = 0.048
- Température = 0.029

Donc, la racine de l'arbre de décision testera l'attribut "Ciel". Ensuite, du fait que l'attribut "Ciel" peut prendre trois valeurs : pour "Ensoleillé", l'algorithme de construction est appelé d'une manière récursive avec les cinq exemples : x1, x2, x8, x9, x11 pour lesquels l'attribut "Ciel" vaut "Ensoleillé" . Les gains d'information des 3 attributs restants sont alors :

- Humidité = 0.970
- Vent = 0.570
- Température = 0.019

L'attribut "Humidité" est donc choisi. La construction de l'arbre de décision est continue d'une manière récursive. Enfin comme résultat nous aurons l'arbre de décision illustré sur la Figure 1.

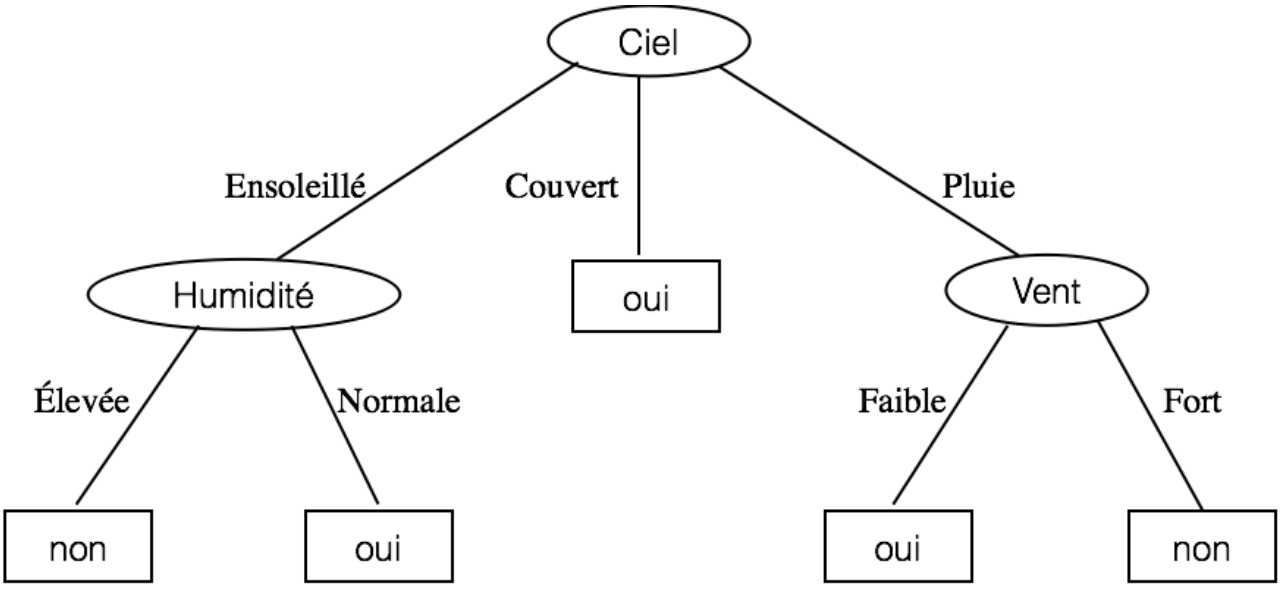

FIGURE B.1 – Exemple de construction d'un arbre de décision.

B.2 Les arbres de décisions REPTree issue de $\mathcal{IUDT}$

REPTree (**DTP**)

==========

sCF-Sigs < 0.13
| sCF-A2s < 0.07
| | sfMXPx-Sigs < 53.5 : 2.000000 (42/4) [25/7]
| | sfMXPx-Sigs >= 53.5 : 1.000000 (7/0) [3/0]
| sCF-A2s >= 0.07
| | sMeanFFT-Sigs < 0
| | | sfMXPx-Sigs < 52.5
| | | | sfMXPx-Sigs < 10.5 : 1.000000 (18/0) [6/0]
| | | | sfMXPx-Sigs >= 10.5
| | | | | sfMXPx-Sigs < 50.5 : 2.000000 (30/2) [12/0]
| | | | | sfMXPx-Sigs >= 50.5 : 1.000000 (18/2) [7/2]
| | | sfMXPx-Sigs >= 52.5
| | | | sCF-A2s < 0.08 : 1.000000 (13/0) [6/2]
| | | | sCF-A2s >= 0.08
| | | | | sRMS-Sigs < 3.03 : 1.000000 (2/0) [2/0]
| | | | | sRMS-Sigs >= 3.03 : 7.000000 (25/0) [16/3]
| | sMeanFFT-Sigs >= 0
| | | sfMXPx-A1s < 48 : 7.000000 (36/0) [17/0]
| | | sfMXPx-A1s >= 48
| | | | sfMXPx-Sigs < 53 : 1.000000 (7/0) [4/0]
| | | | sfMXPx-Sigs >= 53 : 7.000000 (7/0) [3/0]
sCF-Sigs >= 0.13
| sRMS-Sigs < 4.98
| | sRMS-Sigs < 2.81 : 2.000000 (2/0) [1/0]
| | sRMS-Sigs >= 2.81 : 1.000000 (3/2) [3/1]
| sRMS-Sigs >= 4.98 : 3.000000 (70/0) [35/0]

REPTree (**IUDTSD**)

```
================
sCF-Sigs < 0.13
|   sfMXPx-Sigs < 51
|   |   sRMS-Sigs < 7
|   |   |   sRMS-Sigs < 4.57 : 2.000000 (7/0) [0/0]
|   |   |   sRMS-Sigs >= 4.57 : 1.000000 (7/0) [0/0]
|   |   sRMS-Sigs >= 7
|   |   |   sCF-A2s < 0.07 : 2.000000 (15/0) [0/0]
|   |   |   sCF-A2s >= 0.07
|   |   |   |   sMeanFFT-Sigs < 0 : 2.000000 (6/0) [0/0]
|   |   |   |   sMeanFFT-Sigs >= 0 : 7.000000 (10/0) [0/0]
|   sfMXPx-Sigs >= 51
|   |   sfMXPx-A2s < 53.5 : 1.000000 (10/1) [0/0]
|   |   sfMXPx-A2s >= 53.5
|   |   |   sMeanFFT-Sigs < 0 : 1.000000 (6/0) [0/0]
|   |   |   sMeanFFT-Sigs >= 0 : 7.000000 (16/0) [0/0]
sCF-Sigs >= 0.13
|   sRMS-Sigs < 5.11 : 2.000000 (2/1) [0/0]
|   sRMS-Sigs >= 5.11 : 3.000000 (26/0) [0/0]
```

REPTree (**IUDTAS**)

==========

```
sMeanFFT-A2s < 0
|   sRMS-D2s < 1.28 : 2.000000 (15/0) [0/0]
|   sRMS-D2s >= 1.28
|   |   sRMS-D2s < 3.94
|   |   |   sMeanFFT-A2s < 0
|   |   |   |   sRMS-D2s < 1.41
|   |   |   |   |   sMeanFFT-A2s < 0
|   |   |   |   |   |   sMeanFFT-A2s < 0 : 2.000000 (5/1) [0/0]
|   |   |   |   |   |   sMeanFFT-A2s >= 0 : 1.000000 (4/0) [0/0]
|   |   |   |   |   sMeanFFT-A2s >= 0 : 7.000000 (8/0) [0/0]
|   |   |   |   sRMS-D2s >= 1.41
|   |   |   |   |   sMeanFFT-A2s < 0 : 1.000000 (21/0) [0/0]
|   |   |   |   |   sMeanFFT-A2s >= 0
|   |   |   |   |   |   sMean-Freqdsit-A1s < 10.83 : 7.0000 (2/0) [0/0]
|   |   |   |   |   |   sMean-Freqdsit-A1s >= 10.83 : 1.000 (8/0) [0/0]
|   |   |   sMeanFFT-A2s >= 0 : 7.000000 (24/1) [0/0]
|   |   sRMS-D2s >= 3.94
|   |   |   sMean-Freqdsit-A1s < 10.17 : 1.000000 (19/0) [0/0]
|   |   |   sMean-Freqdsit-A1s >= 10.17 : 2.000000 (14/1) [0/0]
sMeanFFT-A2s >= 0
|   sMeanFFT-A2s < 0.01
|   |   sCF-D1s < 0.14
|   |   |   sRMS-D2s < 15.12 : 7.000000 (17/0) [0/0]
|   |   |   sRMS-D2s >= 15.12 : 3.000000 (2/0) [0/0]
|   |   sCF-D1s >= 0.14
|   |   |   sCF-D1s < 0.17 : 3.000000 (10/1) [0/0]
|   |   |   sCF-D1s >= 0.17 : 3.000000 (39/0) [0/0]
|   sMeanFFT-A2s >= 0.01 : 2.000000 (22/0) [0/0]
```

REPTree $\mathcal{IUDT}$

==========

```
sSkew-Sigs < 0.23
|   sMeanFFT-Sigs < 0.01
|   |   sMeanFFT-Sigs < 0
|   |   |   sRMS-A1s < 2.18 : 2.000000 (9/0) [0/0]
|   |   |   sRMS-A1s >= 2.18
|   |   |   |   sRMS-A1s < 8.46
|   |   |   |   |   sMeanFFT-Sigs < 0
|   |   |   |   |   |   sMeanFFT-Sigs < 0 : 1.000000 (9/0) [0/0]
|   |   |   |   |   |   sMeanFFT-Sigs >= 0
|   |   |   |   |   |   |   sMean-Freqdsit-Sigs < 11.83 : 7.00000 (7/0) [0/0]
|   |   |   |   |   |   |   sMean-Freqdsit-Sigs >= 11.83 : 1.0000 (8/0) [0/0]
|   |   |   |   |   sMeanFFT-Sigs >= 0 : 7.000000 (10/0) [0/0]
|   |   |   |   sRMS-A1s >= 8.46
|   |   |   |   |   sMean-Freqdsit-Sigs < 9.67 : 1.000000 (8/0) [0/0]
|   |   |   |   |   sMean-Freqdsit-Sigs >= 9.67 : 2.000000 (6/0) [0/0]
|   |   sMeanFFT-Sigs >= 0
|   |   |   sCF-D1s < 0.15 : 7.000000 (11/0) [0/0]
|   |   |   sCF-D1s >= 0.15 : 3.000000 (5/1) [0/0]
|   sMeanFFT-Sigs >= 0.01 : 2.000000 (13/0) [0/0]
sSkew-Sigs >= 0.23 : 3.000000 (19/0) [0/0]
```

Bibliographie

[Agrawal *et al.*1994] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[Bermejo *et al.*2012] Pablo Bermejo, Luis de la Ossa, José A Gámez, and José M Puerta. Fast wrapper feature subset selection in high-dimensional datasets by means of filter re-ranking. *Knowledge-Based Systems*, 25(1) :35–44, 2012.

[Blake and Merz1998] Catherine L Blake and Christopher J Merz. Uci repository of machine learning databases [http ://www. ics. uci. edu/˜ mlearn/mlrepository. html]. irvine, ca : University of california. *Department of Information and Computer Science*, 460, 1998.

[Boley *et al.*2010] Mario Boley, Tamás Horváth, Axel Poigné, and Stefan Wrobel. Listing closed sets of strongly accessible set systems with applications to data mining. *Theoretical computer science*, 411(3) :691–700, 2010.

[Bourrillion and others2010] K Bourrillion et al. Guava : Google core libraries for java 1.5+(2010), 2010.

[Bradford *et al.*1998] Jeffrey P Bradford, Clayton Kunz, Ron Kohavi, Cliff Brunk, and Carla E Brodley. Pruning decision trees with misclassification costs. In *Machine Learning : ECML-98*, pages 131–136. Springer, 1998.

[Breiman *et al.*1984] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. wadsworth & brooks. *Monterey, CA*, 1984.

[Cheng *et al.*2006] James Cheng, Yiping Ke, and Wilfred Ng. Graphgen. a graph synthetic generator, 2006.

[Chi *et al.*2005] Yun Chi, Richard R Muntz, Siegfried Nijssen, and Joost N Kok. Frequent subtree mining-an overview. *Fundamenta Informaticae*, 66(1) :161–198, 2005.

[Collins2011] William Collins. *Data structures and the Java collections framework*. Wiley Publishing, 2011.

[Djebala *et al.*2008] Abderrazek Djebala, Nouredine Ouelaa, and Nacer Hamzaoui. Detection of rolling bearing defects using discrete wavelet analysis. *Meccanica*, 43(3) :339–348, 2008.

[Esposito *et al.*1997] Floriana Esposito, Donato Malerba, Giovanni Semeraro, and J Kay. A comparative analysis of methods for pruning decision trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(5) :476–491, 1997.

[Fiedler and Borgelt] Mathias Fiedler and Christian Borgelt. Support computation for mining frequent subgraphs in a single graph.

[Fournier and Crémilleux2002] Dominique Fournier and Bruno Crémilleux. A quality index for decision tree pruning. *Knowledge-Based Systems*, 15(1) :37–43, 2002.

[Fredman and Khachiyan1996] Michael L Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3) :618–628, 1996.

[Ganter and Reuter1991] Bernhard Ganter and Klaus Reuter. Finding all closed sets : A general approach. *Order*, 8(3) :283–290, 1991.

[Garey and Johnson1979] Michael R Garey and David S Johnson. Computers and intractibility, 1979.

[Grahne and Zhu2005] Gosta Grahne and Jianfei Zhu. Fast algorithms for frequent itemset mining using fp-trees. *Knowledge and Data Engineering, IEEE Transactions on*, 17(10) :1347–1362, 2005.

[Gunopulos *et al.*1997] Dimitrios Gunopulos, Heikki Mannila, Roni Khardon, and Hannu Toivonen. Data mining, hypergraph transversals, and machine learning. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 209–216. ACM, 1997.

[Gunopulos *et al.*2003] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems (TODS)*, 28(2) :140–174, 2003.

[Hall *et al.*2009] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software : an update. *ACM SIGKDD Explorations Newsletter*, 11(1) :10–18, 2009.

[Horváth *et al.*2010] Tamás Horváth, Jan Ramon, and Stefan Wrobel. Frequent subgraph mining in outerplanar graphs. *Data Mining and Knowledge Discovery*, 21(3) :472–508, 2010.

[Huan *et al.*2003] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 549–552. IEEE, 2003.

[Huan *et al.*2004] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang. Spin : mining maximal frequent subgraphs from graph databases. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–586. ACM, 2004.

[Inokuchi *et al.*2000] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*, pages 13–23. Springer, 2000.

[Ishibuchi *et al.*2001] Hisao Ishibuchi, Tomoharu Nakashima, and Manabu Nii. Genetic-algorithm-based instance and feature selection. In *Instance Selection and Construction for Data Mining*, pages 95–112. Springer, 2001.

[Kankar *et al.*2011] PK Kankar, Satish C Sharma, and SP Harsha. Fault diagnosis of ball bearings using continuous wavelet transform. *Applied Soft Computing*, 11(2) :2300–2312, 2011.

[Karabadji and Seridi2014a] Nour el islem Karabadji and Hassina Seridi. Découverte des sous graphes connexes fréquents fermés à étiquetage d'arêtes non redondant. In *Colloque sur l'Optimisation et les Systèmes d'Information, 2014. COSI 2014, Proceedings*, 2014.

[Karabadji and Seridi2014b] Nour el islem Karabadji and Hassina Seridi. Maximal connected frequent subgraph mining. *EGC 2014*, 2014.

[Karabadji *et al.*2012a] Nour El Islem Karabadji, Ilyes Khelf, Hassina Seridi, and Lakhdar Laouar. Genetic optimization of decision tree choice for fault diagnosis in an industrial

ventilator. In *Condition Monitoring of Machinery in Non-Stationary Operations*, pages 277–283. Springer, 2012.

[Karabadji *et al.*2012b] Nour El Islem Karabadji, Hassina Seridi, Ilyes Khelf, and Lakhdar Laouar. Decision tree selection in an industrial machine fault diagnostics. In *Model and Data Engineering*, pages 129–140. Springer, 2012.

[Karabadji *et al.*2014] Nour El Islem Karabadji, Hassina Seridi, Ilyes Khelf, Nabiha Azizi, and Ramzi Boulkroune. Improved decision tree construction based on attribute selection and data sampling for fault diagnosis in rotating machines. *Engineering Applications of Artificial Intelligence*, 35 :71–83, 2014.

[Kemmar *et al.*2013] Amina Kemmar, Yahia Lebbah, Samir Loudni, and Mohammed Ouali. Lower and upper queries for graph-mining. In *Int. Workshop Languages for Data Mining and Machine Learning co-located with the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2013)*, pages 95–107, 2013.

[Khelf *et al.*2013] Ilyes Khelf, Lakhdar Laouar, Abdelaziz M Bouchelaghem, Didier Rémond, and Salah Saad. Adaptive fault diagnosis in rotating machines using indicators selection. *Mechanical Systems and Signal Processing*, 40(2) :452–468, 2013.

[KHELF2014] Ilyes KHELF. *Diagnostic des machines tournantes par les techniques de l'intelligence artificielle, Badji Mokhtar University.* PhD thesis, 2014.

[Kohavi and John1997] Ron Kohavi and George H John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1) :273–324, 1997.

[Kotsiantis2013] Sotiris B Kotsiantis. Decision trees : a recent overview. *Artificial Intelligence Review*, pages 1–23, 2013.

[Krishna *et al.*2011] Varun Krishna, NNR Ranga Suri, and G Athithan. A comparative survey of algorithms for frequent subgraph discovery. *CURRENT SCIENCE*, 100(2) :190, 2011.

[Kuramochi and Karypis2001] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 313–320. IEEE, 2001.

[Kuramochi and Karypis2005] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph*. *Data mining and knowledge discovery*, 11(3) :243–271, 2005.

[Liu2010] Huan Liu. *Instance selection and construction for data mining*. Springer-Verlag, 2010.

[Luo *et al.*2013] Linkai Luo, Xiaodong Zhang, Hong Peng, Weihang Lv, and Yan Zhang. A new pruning method for decision tree based on structural risk of leaf node. *Neural Computing and Applications*, pages 1–10, 2013.

[Macaš *et al.*2012] Martin Macaš, Lenka Lhotská, Eduard Bakstein, Daniel Novák, Jiří Wild, Tomáš Sieger, Pavel Vostatek, and Robert Jech. Wrapper feature selection for small sample size data driven by complete error estimates. *Computer methods and programs in biomedicine*, 108(1) :138–150, 2012.

[Mannila and Toivonen1997] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data mining and knowledge discovery*, 1(3) :241–258, 1997.

[Marmion2011] Marie-Éléonore Marmion. *Recherche locale et optimisation combinatoire : de l'analyse structurelle d'un problème à la conception d'algorithmes efficaces*. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2011.

[McKay1981] Brendan D McKay. Practical graph isomorphism, congressus numerantium, 30 (1981) 45–87. *Nauty web page : http ://cs. anu. edu. au/˜ bdm/nauty*, 1981.

[Mingers1987] John Mingers. Expert systems-rule induction with statistical data. *Journal of the operational research society*, pages 39–47, 1987.

[Mingers1989] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine learning*, 4(2) :227–243, 1989.

[Niblett and Bratko1987] Tim Niblett and Ivan Bratko. Learning decision rules in noisy domains. In *Proceedings of Expert Systems' 86, The 6Th Annual Technical Conference on Research and development in expert systems III*, pages 25–34. Cambridge University Press, 1987.

[Nijssen and Kok2003] Siegfried Nijssen and Joost N Kok. Efficient discovery of frequent

unordered trees. In *First international workshop on mining graphs, trees and sequences*, pages 55–64, 2003.

[Nijssen and Kok2005] Siegfried Nijssen and Joost N Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1) :77–87, 2005.

[Nijssen2010a] Siegfried Nijssen. Constraint-based mining. *Encyclopedia of Machine Learning*, pages 221–225, 2010.

[Nijssen2010b] Siegfried Nijssen. Tree mining. *Encyclopedia of Machine Learning*, pages 991–999, 2010.

[Nourine and Petit2012] Lhouari Nourine and Jean-Marc Petit. Extending set-based dualization : Application to pattern mining. In *ECAI*, volume 242, pages 630–635, 2012.

[Piramuthu2008] Selwyn Piramuthu. Input data for decision trees. *Expert Systems with applications*, 34(2) :1220–1226, 2008.

[Plauger *et al.*2000] Phillip James Plauger, Meng Lee, David Musser, and Alexander A Stepanov. *C++ Standard Template Library*. Prentice Hall PTR, 2000.

[Quinlan1987] J. Ross Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3) :221–234, 1987.

[Quinlan1993] John Ross Quinlan. *C4. 5 : programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[Schmidt and Druffel1976] Douglas C Schmidt and Larry E Druffel. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *Journal of the ACM (JACM)*, 23(3) :433–445, 1976.

[Thomas *et al.*2009] Lini Thomas, S Valluri, and Kamalakar Karlapalem. Isg : Itemset based subgraph mining. Technical report, Technical Report, IIIT, Hyderabad, December2009, 2009.

[Thomas *et al.*2010] Lini T Thomas, Satyanarayana R Valluri, and Kamalakar Karlapalem. Margin : Maximal frequent subgraph mining. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(3) :10, 2010.

[Ullmann1976] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1) :31–42, 1976.

[Uno *et al.*2004] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 2 : Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, volume 126, 2004.

[Yan and Han2002] Xifeng Yan and Jiawei Han. gspan : Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 721–724. IEEE, 2002.

[Yan and Han2003] Xifeng Yan and Jiawei Han. Closegraph : mining closed frequent graph patterns. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 286–295. ACM, 2003.