

Gradualité et bases multidimensionnelles

Lisa Di Jorio

Rapport de Recherche

1 Introduction

Le concept des entrepôts de données est apparu dans les années 1990. Ils permettent de stocker des masses de données historisées provenant de diverses sources dans une structure adaptée. Ces données sont ensuite utilisables sur demande des utilisateurs et chargées selon différents formats. Par exemple, le cube de données vise à présenter les données dans un format adapté aux décideurs, qui pourront naviguer à différents niveaux de granularité. La fouille de cube consiste à extraire de nouveaux types de connaissances telles que les règles d'associations multidimensionnelles, les motifs séquentiels multidimensionnels, ou encore plus récemment les blocs de données multidimensionnels. Cependant, il n'existe actuellement aucune méthode permettant d'extraire des notions graduelles à partir de telles structures. Pourtant, la plupart des données médicales sont multidimensionnelles par nature, par exemple parce qu'elles décrivent un résultat biologique ou médical en fonction de différents axes d'analyse comme l'âge, le sexe ou la ville d'habitation. Différents niveaux de hiérarchie sont souvent décrits (comme par exemple ville, département, région, ou encore les familles de maladies). De plus, les cubes de données présentent l'avantage de conserver des données numériques, sous un format agrégé.

Dans ce chapitre, nous nous intéressons à la découverte de règles d'association graduelles multidimensionnelles. Notre méthode s'appuie sur l'extraction de blocs de données. Nous résumons dans la section 2 les différentes notions associées aux cubes de données, ainsi que les différents travaux de fouille de données basés sur les cubes. Dans la section 3, nous présentons un algorithme efficace d'extraction de blocs de données. Ces blocs sont la base de notre algorithme d'extraction de règles d'association graduelles multidimensionnelles, présenté dans la section 4. Enfin, ce chapitre se termine par une discussion.

2 État de l'art

2.1 Cubes de données et fouille de données

Un cube de données est structuré par une ou plusieurs dimensions qui peuvent être numériques. Sur ces dimensions, il est également possible de définir une hiérarchie, ce qui permet d'obtenir une vision plus ou moins précise des données. Un cube est composé de cellules, où chaque cellule correspond à la valeur prise à l'intersection des membres des dimensions. Les valeurs contenues dans ces cellules sont alors une *mesure* agrégée en regroupant des valeurs de données sources sur chaque dimension. La figure 1 illustre un tel cube. Il est structuré par trois dimensions : les gènes, l'âge et le grade de la tumeur.

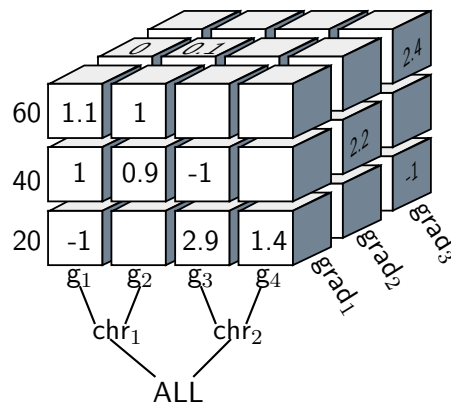


FIG 1 – Exemple de cube de données

Une hiérarchie est établie sur les gènes, qui peuvent être regroupés par chromosomes. Enfin, le contenu des cellules représente la moyenne des expressions de gènes pour tous les patients de la base associés à ces valeurs de dimension. Ainsi, la cellule en bas à droite signifie que la moyenne d'expression du gène g_1 pour tous les patients ayant 20 ans et une tumeur de grade 1 est de 1.4.

La fouille de cube de données consiste à définir des méthodes capables d'extraire des connaissances à partir de données multidimensionnelles, agrégées, et potentiellement organisées à différents niveaux de hiérarchies. Différente de la fouille de données classique en raison des spécificités propres à cette organisation des données, elle nécessite la définition de nouvelles méthodes permettant à la fois d'extraire des connaissances intéressantes et pertinentes, mais aussi de faire face à de gros volumes de données en raison de la taille sans cesse croissante des cubes de données disponibles et des besoins grandissants d'applications en quasi temps réel des utilisateurs.

Dans ce contexte, de nombreux travaux ont été proposés ces dernières années, notamment pour extraire des règles d'association [KHC97, IKA02], des résumés flous [Lau02], ou encore des motifs séquentiels multidimensionnels [PHP⁺01, PCL⁺05]. Il est alors possible d'extraire des règles du type "la plupart des gènes g_1 des tumeurs de grade 1 s'expriment faiblement", ou "la plupart des gènes g_1 faiblement exprimés se retrouvent dans les tumeurs de grade 1 ou encore "pour la plupart des tumeurs de grade 3, on trouve de fortes expressions dans le chromosome 1 pour les patients de 20 ans, puis des expressions de gènes g_3 faibles dans les tumeurs de grade 3".

Ces différentes règles, profitant de l'organisation des cubes de données, de la présence d'une ou plusieurs mesures et de leur multidimensionnalité et organisation multi-niveaux, permettent de renseigner le décideur sur les tendances présentes dans les cubes de données. Ces tendances sont difficiles à retrouver par une simple navigation non guidée par les opérateurs classiques OLAP. Notons que des travaux ont également été proposés pour retrouver des exceptions au sein de telles données [PGG⁺07, PLT07].

Cependant, il n'existe pas à notre connaissance de méthode permettant d'extraire des règles générales de la forme "Plus l'âge et le grade de la tumeur sont élevés, plus l'expression moyenne augmente".

Or si l'ensemble des dimensions des cubes de données manipulés n'est pas toujours ordonné, il n'en reste pas moins que beaucoup peuvent l'être, notamment en observant les hiérarchies définies, comme par exemple les gènes selon leur chromosomes. Ce type de règles permet alors de retrouver des corrélations au sein de ces dimensions ordonnées.

Toutefois, les cubes de données étant volumineux et contenant des valeurs souvent très agrégées, il n'est pas possible de considérer chaque cellule individuellement pour vérifier si la corrélation est respectée. Afin de prendre en compte de manière plus souple les valeurs présentes dans les cellules du cube, nous proposons donc de nous appuyer sur une représentation des cubes de données en blocs, comme proposé dans [CLL08, CMLL04, CLL07]. Ces blocs de données représentent des zones du cube quasi homogènes (au sens d'une confiance), décrivant par exemple que *quand la ville est N.Y. ou S.F. et que le produit est P1 ou P3 alors le niveau de ventes est 4*. Ces cubes de données, extraits par des algorithmes par niveau, présupposent qu'une représentation du cube a été définie. Une représentation correspond intuitivement à une façon d'agencer les valeurs des dimensions (en plaçant par exemple le produit *P3* puis *P1* puis *P4* puis *P2*). Dans le cadre de dimensions ordonnées, cette représentation sera obtenue en ordonnant les dimensions.

2.2 Blocs de données

L'extraction de règles graduelles à partir de cubes se trouve au carrefour de plusieurs problématiques : l'extraction de connaissances au sein de cubes multidimensionnels (règles d'association et motifs séquentiels), la présentation des connaissances contenues dans le cube à l'utilisateur [CLM03], ou encore la prise en compte de la mesure [PLT07] et des hiérarchies [CLL07].

Dans [CLM03], les auteurs mettent en évidence qu'il peut exister plusieurs représentations équivalentes pour des cubes définis sur plusieurs dimensions. Il suffit d'inverser l'ordre de présentation des membres d'une dimension pour obtenir une nouvelle représentation d'un même cube. Les auteurs montrent également qu'il existe des représentations plus pertinentes que d'autres selon des critères définis par l'utilisateur. Par exemple, une représentation pertinente peut être basée sur la mesure : on peut imaginer un ordre de présentation de la mesure dont on retrouverait les plus faibles valeurs à un *coin* du cube et les plus fortes au *coin opposé*. Les tableaux 1a et 1b montrent de telles représentations : les valeurs des cellules ne changent pas, mais des inversions sur les deux dimensions du cube réordonnent les cellules en plaçant les valeurs les plus faibles en bas à gauche et les plus fortes en haut à droite.

Les auteurs discernent les *représentations parfaites*, pour lesquelles aucune cellule ne contredit la contrainte fournie par l'utilisateur (l'ordre par exemple), des *représentations optimales*, qui contiennent des cellules contradictoires, mais qu'il est impossible de contourner. Le but est alors de calculer les meilleures représentations, en utilisant les opérateurs OLAP d'inversion. Cependant, le problème est NP-complet, ce qui rend la recherche de telles représentations difficile. De plus, même si la lecture du cube est facilitée pour l'utilisateur, il se peut que le fait de "casser" l'ordre entre certains membres d'une dimension rende l'interprétation difficile.

Dans [CLL07], les auteurs considèrent qu'il existe plusieurs représentations, mais ils en supposent une choisie (par l'utilisateur par exemple) comme support à la fouille. Il s'agit alors d'extraire des *blocs*

| Patients avec gènes sur-exprimés | | | | | |
|----------------------------------|--------|--------|--------|--------|--------|
| gène 4 | 3 | 5 | 6 | 3 | 5 |
| gène 3 | 4 | 6 | 7 | 5 | 7 |
| gène 2 | 2 | 4 | 6 | 2 | 5 |
| gène 1 | 4 | 5 | 7 | 4 | 6 |
| | 20 ans | 30 ans | 40 ans | 50 ans | 60 ans |

(a)

| Patients avec gènes sur-exprimés | | | | | |
|----------------------------------|--------|--------|--------|--------|--------|
| gène 3 | 4 | 5 | 6 | 7 | 7 |
| gène 1 | 4 | 4 | 5 | 6 | 7 |
| gène 4 | 3 | 3 | 5 | 5 | 6 |
| gène 2 | 2 | 2 | 4 | 5 | 6 |
| | 20 ans | 50 ans | 30 ans | 60 ans | 40 ans |

(b)

TAB 1 – Deux représentations différentes pour un cube de données

permettant de dériver des règles de la forme “Si les produits sont P_1, P_2 et P_3 et les mois d’achat sont juillet et janvier, alors le nombre moyen d’achats est 800”. Une version améliorée de l’algorithme est présentée dans [CLL08] et permet de prendre en compte la hiérarchie des dimensions. Nous présentons ici brièvement la méthode de [CLL07], ainsi que les formalisation associées. Dans la suite de ce chapitre, nous conserverons ces notations.

Les cubes de données sont définis de manières diverses. Dans ce chapitre, nous considérons un ensemble de dimensions $D = \{d_1, d_2, \dots, d_n\}$ où chaque dimension d_i est définie sur un domaine fini de valeurs noté dom_i . Un cube de données est alors défini à partir de ces dimensions.

Définition 1. (Cube) *Un cube k -dimensionnel, ou simplement un cube C est un n -uplet*

$\langle dom_1, \dots, dom_k, dom_{mes}, m_C \rangle$ où :

- dom_1, \dots, dom_k sont des ensembles finis de symboles pour les membres associés avec les dimensions d_1, \dots, d_k respectivement
- dom_{mes} un ensemble fini et totalement ordonné de valeurs de mesure. Soit $\perp \notin dom_{mes}$ une constante (pour représenter les valeurs nulles). Alors $dom_m = dom_{mes} \cup \perp$
- m_C est une application $m_c : dom_1 \times \dots \times dom_k \rightarrow dom_m$ où m est le domaine de la mesure.

Pour chaque $i = 1, \dots, k$, un élément v_i dans dom_i est appelé une *valeur membre*. Une cellule c d’un cube k -dimensionnel C est un $(k + 1)$ -uplet $\langle v_1, \dots, v_k, m \rangle$ tel que pour tout $i = 1, \dots, k$, v_i appartient à dom_i et $m = m_C(v_1, \dots, v_k)$. m est appelé le *contenu* de c .

Par exemple, le tableau 2a illustre un cube à deux dimensions. Nous avons $C = \langle \{1, 2, 3, 4, 5\}, \{10, 20, 30, 40, 50\}, \{4, 5, 6, 8\}, m_C \rangle$. Nous avons deux dimensions $d_1 =$ Stade de la maladie et $d_2 =$ Nombre de jours, avec $dom_1 = \{1, 2, 3, 4, 5\}$ et $dom_2 = \{10, 20, 30, 40, 50\}$. 10 est un membre de dom_2 .

De manière générale, les cubes de données sont présentés à l’utilisateur sur une ou deux dimensions.

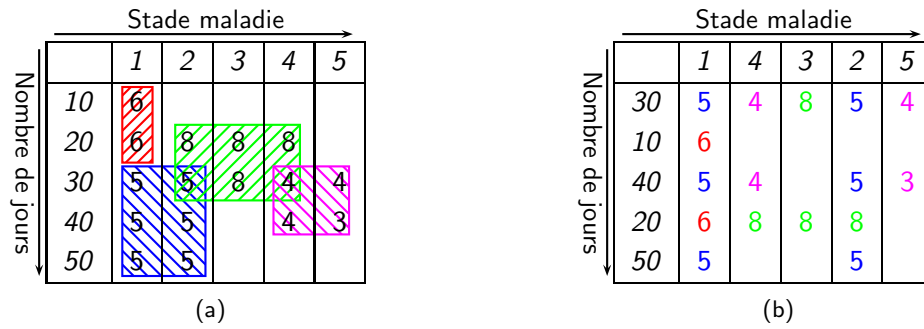


FIG 2 – (a) Exemple de blocs de données (b) Représentation différente du même cube

Les membres des dimensions sont affichés par ordre alphabétique ou ordre d'insertion dans la base et les dimensions dont les membres peuvent être ordonnés (dimension temporelle par exemple) sont affichées de manière ordonnée si insérées dans cet ordre. Cependant, cette représentation est arbitraire. En effet, il est possible d'afficher les membres dans un ordre différent. Les mêmes données seront alors présentées à l'utilisateur différemment. Par exemple, nous aurions pu afficher la figure 2a sous la représentation de la figure 2b, où l'ordre des lignes a été modifié (ainsi que celui des colonnes). Il existe donc différentes représentations d'un même cube, qui sont formalisées de la manière suivante :

Définition 2. (Représentation) Une représentation d'un cube k -dimensionnel C est un ensemble $R = \{rep_1, \dots, rep_k\}$ où pour chaque $i = 1, \dots, k$, rep_i est une application injective de dom_i vers $\{1, \dots, |dom_i|\}$

Dans ce chapitre, nous considérons comme dans [CLL07] une représentation donnée. Cette représentation peut avoir été fixée par l'utilisateur ou provenir d'opérations antérieures. Nous ne traitons pas cet aspect dans ce chapitre. A partir de telles représentations, il est alors possible de retrouver les zones homogènes (au sens de la valeur de la mesure). Pour ce faire, [CLL07] proposent d'extraire des blocs de données, qui sont définis de la manière suivante :

Définition 3. (Bloc) Un bloc b est un ensemble de cellules définies sur un cube k -dimensionnel C par $b = \delta_1 \times \dots \times \delta_k$ où les δ_i sont des intervalles de valeurs consécutives de dom_i , pour $i = 1, \dots, k$.

Remarque : il est possible que δ_i soit égal à tout dom_i (valeur notée ici ALL_i).

Tout comme les règles d'association, il est possible de définir des mesures de fréquence et de confiance pour les blocs :

Définition 4. (Fréquence) Soit $count(b, m)$ le nombre de cellules ayant la valeur m dans b , alors la fréquence d'un bloc b de C pour une valeur de mesure m est

$$Freq(b) = \frac{count(b, m)}{|C|}$$

Définition 5. (Confiance) La confiance d'un bloc b pour une mesure m est

$$Conf(b) = \frac{count(b, m)}{|b|}$$

Par exemple, il est possible d'extraire quatre blocs à partir du cube de la figure 2a :

- $b_1 = [1, 2] \times [30, 50]$, associé à la valeur 5, avec $Freq(b_1) = \frac{6}{25} = 0.24$ et $Conf(b_1) = \frac{6}{6} = 1$
- $b_2 = [2, 4] \times [20, 30]$, associé à la valeur 8, avec $Freq(b_2) = \frac{6}{25} = 0.24$ et $Conf(b_2) = \frac{6}{4} \simeq 0.66$
- $b_3 = [4, 5] \times [30, 40]$, associé à la valeur 4, avec $Freq(b_3) = \frac{4}{25} = 0.16$ et $Conf(b_3) = \frac{3}{4} = .075$
- $b_4 = [1, 1] \times [10, 20]$, associé à la valeur 6, avec $Freq(b_4) = \frac{2}{25} = 0.08$ et $Conf(b_4) = \frac{2}{2} = 1$

A partir de ces blocs, des règles “sémantiques” peuvent être proposées à l'utilisateur. Par exemple b_1 peut être formulé de la manière suivante : “pour un nombre de jours compris entre 30 et 50 et pour chaque stade de maladie de 1 et 2, il y a 5 patients atteints”.

[CLL07] propose une méthode basée sur une génération par niveau afin d'extraire de tel blocs. Voici les principales étapes de l'algorithme, effectuées pour chaque mesure m présente dans le cube :

1. Pour chaque dimension d_i , toutes les tranches contiguës sont générées. Informellement, une tranche est un hypercube ne contenant qu'un membre de $v_i \in d_i$ et ALL_j sur toutes les autres dimensions. Par exemple, pour le cube de la figure 1, il y a trois tranches pour la dimension “âge” : $(ALL_{gene} \times 20 \times ALL_{grad})$, $(ALL_{gene} \times 40 \times ALL_{grad})$ et $(ALL_{gene} \times 60 \times ALL_{grad})$.
2. Pour chaque tranche, les intervalles des dimensions fixées à ALL sont raffinées afin de minimiser la taille des blocs de données et d'en affiner la qualité. Puis, toutes les tranches telles que $Freq(\mathcal{T}(v_i), m) > \sigma$ sont conservées dans un ensemble \mathcal{L}_1 .
3. Pour chaque dimension, les membres apparaissant dans chaque intervalle sont extraits et stockés sous la forme (d_i, v) .
4. Pour chaque dimension, les intervalles maximaux sont calculés en fonction des couples précédents.
5. L'algorithme entre ensuite dans une phase “à la apriori”. Il s'agit d'augmenter la taille des blocs dimension par dimension à chaque passe, en combinant les blocs trouvés à la passe précédente. L'algorithme s'arrête lorsqu'il n'y a plus de blocs fréquents.

Afin d'illustrer cet algorithme, nous le déroulons pour la mesure 8 et $\sigma = 5$ (les blocs fréquents doivent contenir au moins 5 cellules). Le tableau 2 résume ces différentes étapes. Tout d'abord, les tranches sont construites. Pour plus de lisibilité, nous avons regroupé les tranches par dimension. Notons que quelle que soit la mesure et le seuil de fréquence minimal considéré, ces tranches seront les mêmes. A l'issue de cette étape, 10 tranches sont construites. Elles ont toutes la même fréquence, soit 5 cellules. Ensuite, ces tranches sont raffinées pour calculer les intervalles de cellules contenant la valeur 8. Les tranches sur les membres $(d_1, 1)$, $(d_1, 5)$, $(d_2, 10)$, $(d_2, 40)$ et $(d_2, 50)$ sont élaguées, car elles ne contiennent pas de cellules ayant la valeur 8. En revanche, 5 tranches sont conservées, chacune ayant des intervalles différents. Au pas 3, les membres des dimensions fréquemment présentes sont isolés, puis combinés au pas 4. A ce stade, pour la valeur 8, il ne reste qu'un intervalle par dimension. Lors du pas 5, ces deux intervalles sont combinés et permettent d'extraire le bloc b_2 . Cette méthode est répétée pour chaque mesure présente dans le cube.

2.3 Discussion

L'algorithme présenté dans [CLL07] n'est pas complet. D'une part, l'élagage basé sur la mesure de confiance en utilisant un algorithme par niveau peut écarter des candidats respectant les seuils de support

| Mesure | Dimensions | |
|------------------------------------------------------------------|--------------------------------------|-----------------------------|
| Pas 1 : construire les tranches pour chaque dimension | | |
| | $[1, 1] \times ALL_{d_2}$ | $ALL_{d_1} \times [10, 10]$ |
| | $[2, 2] \times ALL_{d_2}$ | $ALL_{d_1} \times [20, 20]$ |
| | $[3, 3] \times ALL_{d_2}$ | $ALL_{d_1} \times [30, 30]$ |
| | $[4, 4] \times ALL_{d_2}$ | $ALL_{d_1} \times [40, 40]$ |
| | $[5, 5] \times ALL_{d_2}$ | $ALL_{d_1} \times [50, 50]$ |
| Pas 2 : raffiner les tranches | | |
| 8 | $[2, 2] \times [20, 20]$ | $[2, 4] \times [20, 20]$ |
| 8 | $[3, 3] \times [20, 30]$ | $[3, 3] \times [30, 30]$ |
| 8 | $[4, 4] \times [20, 20]$ | |
| Pas 3 : sélectionner les membres fréquents pour chaque dimension | | |
| 8 | $(d_1, 2)$ | $(d_2, 20)$ |
| 8 | $(d_1, 3)$ | $(d_2, 30)$ |
| 8 | $(d_1, 4)$ | |
| Pas 4 : Calculer les intervalles maximaux par dimension | | |
| 8 | $[2, 4]$ | $[20, 30]$ |
| Pas 5 : Génération des blocs | | |
| 8 | $b_2 = [2, 4] \times [20, 30]$ | |
| 8 | $Freq(b_2) = 0.24, Conf(b_2) = 0.66$ | |

TAB 2 – Déroulement de l’algorithme pour $m = 8$ et $\sigma = 5$

et confiance minimale à l’étape de génération suivante. D’autre part, la maximisation des intervalles à l’étape 4 peut mener à ne pas considérer des blocs fréquents aux étapes ultérieures. Un exemple détaillé est donné dans [CLL08].

Cependant, l’utilisation des représentations et blocs de données s’avère intéressante, car ces techniques permettent un “remodelage” du cube afin de faire émerger les corrélations de valeurs. De plus, les blocs proposent une perspective intéressante afin d’éliminer le bruit car une mesure de support et de confiance est associée à chaque bloc. Ainsi, les cellules vides font baisser ces mesures, ce qui permet de raffiner les parties du cube sélectionnées jusqu’à n’obtenir que les parties pertinentes. Dans ce chapitre, nous optons donc pour une définition des règles d’associations graduelles basées sur les blocs de données. Nous pensons que la gradualité est une valeur ajoutée forte, puisqu’elle met en évidence la corrélation de variations qu’il existe d’une part entre les membres d’une même dimension, et d’autre part entre les valeurs des blocs, augmentant ainsi la sémantique associée à un bloc. Enfin, il est possible de comparer les valeurs des différents blocs définis sur ces dimensions. Cela place la méthode graduelle comme un complément à la méthode de fouille de cube présentée ci-dessus.

Les données médicales sont multidimensionnelles par nature, et plus particulièrement dans le cas de données cliniques, où chaque attribut (taux de cholestérol, âge, médication) peut être vu comme une dimension [PJ98, PL08]. Les premières expérimentations menées avec les implémentations

proposées dans [CLL07] n'ont au préalable pas permis d'extraire de blocs de données, pour des raisons de performances. Dans ce chapitre, nous répondons aux questions suivantes : *est-il possible de proposer un algorithme d'extraction de bloc de données robuste sur des données médicales ? Comment utiliser les blocs de données afin d'extraire des règles graduelles multidimensionnelles ? Et enfin, est-il possible d'extraire de telles règles à la volée, c'est-à-dire au fur et à mesure de la découverte des blocs ?*

3 Extraction de blocs dans les bases de données multidimensionnelles

Dans cette section, nous présentons un nouvel algorithme d'extraction de blocs. Celui-ci présente l'avantage d'être complet, et nous permet d'extraire des blocs de données à partir de nos bases médicales. Nous proposons une extraction en deux étapes. Dans un premier temps, nous fabriquons des *classes* de blocs. Ces classes sont définies en fonction de la taille des intervalles et permettent de connaître facilement le support associé à chaque bloc. De plus, certaines de ces classes sont incluses dans d'autres. Il est ainsi possible de générer par jointure tous les blocs multidimensionnels d'une classe à partir d'une autre classe. Cela nous permet de décider d'un *plan d'exécution*, c'est-à-dire de décider quelle classe sera utilisée afin de fabriquer tous les blocs de la classe suivante.

Afin d'illustrer notre propos, nous utilisons un exemple fourni par [CLL07], illustré par le tableau 3.

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| P_1 | 6 | 6 | 8 | 5 | 5 | 2 |
| P_2 | 6 | 8 | 5 | 5 | 6 | 75 |
| P_3 | 8 | 5 | 5 | 2 | 2 | 8 |
| P_4 | 8 | 8 | 8 | 2 | 2 | 2 |
| | V_1 | V_2 | V_3 | V_4 | V_5 | V_6 |

TAB 3 – Base exemple à deux dimensions

Cette base est définie sur deux dimensions : $dom(d_1) = \{V_1, \dots, V_6\}$ et $dom(d_2) = \{P_1, \dots, P_4\}$.

3.1 Définition

A partir d'un cube de données, il est aisé de calculer pour chacune des dimensions tous les intervalles possibles. Notons $mathcal{I}_d$ l'ensemble des intervalles possibles associé à la dimension d . Pour d contenant $|dom(d)|$ membres, le nombre d'intervalles est de :

$$|\mathcal{I}_d| = \frac{|dom(d)|(|dom(d)| + 1)}{2}$$

Par exemple, nous pouvons construire 10 intervalles avec les membres de la dimension d_2 , référencés dans le tableau 4. De plus, il est possible de regrouper ces intervalles en fonction de leur taille. Par exemple, pour la dimension d_2 , il y a 4 intervalles de taille 1, 3 intervalles de taille 2, 2 intervalles de taille 3 et 1 intervalle de taille 4.

A partir de ces informations, nous pouvons décrire de manière générale l'ensemble des blocs composant la base en combinant les différentes longueurs d'intervalles. Afin de mieux gérer l'ensemble de ces blocs, nous définissons l'équivalence entre blocs, ce qui nous permet de ranger ces blocs par classe. Une classe est de la forme $\langle i_1 \rangle \times \langle i_2 \rangle \times \dots \times \langle i_n \rangle$, où chaque i_j est une taille d'intervalle.

| Taille | Intervalle | Taille | Intervalle |
|--------|--------------|--------|--------------|
| 1 | $[P_1, P_1]$ | 2 | $[P_2, P_3]$ |
| 1 | $[P_2, P_2]$ | 2 | $[P_3, P_4]$ |
| 1 | $[P_3, P_3]$ | 3 | $[P_1, P_3]$ |
| 1 | $[P_4, P_4]$ | 3 | $[P_2, P_4]$ |
| 2 | $[P_1, P_2]$ | 4 | $[P_1, P_4]$ |

TAB 4 – L'ensemble des intervalles associé à d_2

Définition 6. (*Équivalence entre bloc*) Soient $b = \delta_1 \times \dots \times \delta_k$ et $b' = \delta'_1 \times \dots \times \delta'_k$ deux blocs. On dit que b et b' sont équivalents et on note $b \equiv b'$ si $\forall i \in \{1, \dots, k\}, |\delta_i| = |\delta'_i|$. Comme toute relation d'équivalence, nous avons les propriétés suivantes :

- réflexivité ($b \equiv b$)
- symétrie ($b \equiv b' \Leftrightarrow b' \equiv b$)
- transitivité ($b \equiv b' \wedge b' \equiv b'' \Rightarrow b \equiv b''$)

La définition 6 permet de formaliser les classes d'équivalence :

Définition 7. (*Classe d'équivalence d'un bloc*) Soit \mathcal{B} l'ensemble des blocs d'un cube. La classe d'équivalence d'un bloc $b \in \mathcal{B}$, notée $Cl(b)$ est l'ensemble des images de x par la relation \equiv :

$$Cl(b) = \{b' \in \mathcal{B} | b \equiv b'\}$$

Dans un cube de données contenant k dimensions, le nombre de classes d'équivalences est de :

$$\prod_{i=0}^k |dom_i|$$

Par exemple, pour la base du tableau 3, nous pouvons construire $6 \times 4 = 24$ classes différentes, affichées dans le tableau 5. Notons que la dernière classe étant définie sur la taille maximale de tous les intervalles, elle représentera toujours la base complète.

A chacune de ces classes est associé un certain nombre de blocs, qu'il est possible de calculer à l'avance. Par exemple, la classe $Cl_{12} = \langle 3 \rangle \times \langle 4 \rangle$ contiendra les quatre blocs suivants :

- $b_{12} = [V_1, V_3] \times [P_1, P_4]$, support $12/24 = 0.5$
- $b_{122} = [V_2, V_4] \times [P_1, P_4]$, support $12/24 = 0.5$
- $b_{123} = [V_3, V_5] \times [P_1, P_4]$, support $12/24 = 0.5$
- $b_{124} = [V_4, V_6] \times [P_1, P_4]$, support $12/24 = 0.5$

La génération des blocs de données se fait en utilisant l'opérateur union. L'union de deux blocs est définie à partir de l'union d'intervalles suivante :

Définition 8. (*Union d'intervalles*) Soient δ et δ' deux intervalles. L'union de δ et δ' , notée $\delta \cup \delta'$ est définie par $[\min(\delta, \delta'), \max(\delta, \delta')]$

| Classe | Exemple | Classe | Exemple |
|--------------------------------------------------------|-----------------------------------------|--------------------------------------------------------|-----------------------------------------|
| $Cl_1 = \langle 1 \rangle \times \langle 1 \rangle$ | $b_1 = [V_1, V_1] \times [P_1, P_1]$ | $Cl_{13} = \langle 4 \rangle \times \langle 1 \rangle$ | $b_{13} = [V_1, V_4] \times [P_1, P_1]$ |
| $Cl_2 = \langle 1 \rangle \times \langle 2 \rangle$ | $b_2 = [V_1, V_1] \times [P_1, P_2]$ | $Cl_{14} = \langle 4 \rangle \times \langle 2 \rangle$ | $b_{14} = [V_1, V_4] \times [P_1, P_2]$ |
| $Cl_3 = \langle 1 \rangle \times \langle 3 \rangle$ | $b_3 = [V_1, V_1] \times [P_1, P_3]$ | $Cl_{15} = \langle 4 \rangle \times \langle 3 \rangle$ | $b_{15} = [V_1, V_4] \times [P_1, P_3]$ |
| $Cl_4 = \langle 1 \rangle \times \langle 4 \rangle$ | $b_4 = [V_1, V_1] \times [P_1, P_4]$ | $Cl_{16} = \langle 4 \rangle \times \langle 4 \rangle$ | $b_{16} = [V_1, V_4] \times [P_1, P_4]$ |
| $Cl_5 = \langle 2 \rangle \times \langle 1 \rangle$ | $b_5 = [V_1, V_2] \times [P_1, P_1]$ | $Cl_{17} = \langle 5 \rangle \times \langle 1 \rangle$ | $b_{17} = [V_1, V_5] \times [P_1, P_1]$ |
| $Cl_6 = \langle 2 \rangle \times \langle 2 \rangle$ | $b_6 = [V_1, V_2] \times [P_1, P_2]$ | $Cl_{18} = \langle 5 \rangle \times \langle 2 \rangle$ | $b_{18} = [V_1, V_5] \times [P_1, P_2]$ |
| $Cl_7 = \langle 2 \rangle \times \langle 3 \rangle$ | $b_7 = [V_1, V_2] \times [P_1, P_3]$ | $Cl_{19} = \langle 5 \rangle \times \langle 3 \rangle$ | $b_{19} = [V_1, V_5] \times [P_1, P_3]$ |
| $Cl_8 = \langle 2 \rangle \times \langle 4 \rangle$ | $b_8 = [V_1, V_2] \times [P_1, P_4]$ | $Cl_{20} = \langle 5 \rangle \times \langle 4 \rangle$ | $b_{20} = [V_1, V_5] \times [P_1, P_4]$ |
| $Cl_9 = \langle 3 \rangle \times \langle 1 \rangle$ | $b_9 = [V_1, V_3] \times [P_1, P_1]$ | $Cl_{21} = \langle 6 \rangle \times \langle 1 \rangle$ | $b_{21} = [V_1, V_6] \times [P_1, P_1]$ |
| $Cl_{10} = \langle 3 \rangle \times \langle 2 \rangle$ | $b_{10} = [V_1, V_3] \times [P_1, P_2]$ | $Cl_{22} = \langle 6 \rangle \times \langle 2 \rangle$ | $b_{22} = [V_1, V_6] \times [P_1, P_2]$ |
| $Cl_{11} = \langle 3 \rangle \times \langle 3 \rangle$ | $b_{11} = [V_1, V_3] \times [P_1, P_3]$ | $Cl_{23} = \langle 6 \rangle \times \langle 3 \rangle$ | $b_{23} = [V_1, V_6] \times [P_1, P_3]$ |
| $Cl_{12} = \langle 3 \rangle \times \langle 4 \rangle$ | $b_{12} = [V_1, V_3] \times [P_1, P_4]$ | $Cl_{24} = \langle 6 \rangle \times \langle 4 \rangle$ | $b_{24} = [V_1, V_6] \times [P_1, P_4]$ |

TAB 5 – Classes d'équivalence de blocs construites à partir du cube exemple

Définition 9. (*Union de blocs*) Soient $b = \delta_1 \times \dots \times \delta_k$ et $b' = \delta'_1 \times \dots \times \delta'_k$ deux blocs. L'union de b et b' , notée $b \cup b'$ est définie par $\delta_1 \cup \delta'_1 \times \dots \times \delta_k \cup \delta'_k$.

Par exemple, si l'on unit les blocs b_{12}, b_{122}, b_{123} et b_{124} , nous obtenons les blocs suivants :

- $b_{12} \cup b_{122} = [V_1, V_4] \times [P_1, P_4] = b_{16}$
- $b_{122} \cup b_{123} = [V_2, V_5] \times [P_1, P_4] = b_{161}$
- $b_{123} \cup b_{124} = [V_3, V_6] \times [P_1, P_4] = b_{162}$
- $b_{12} \cup b_{123} = [V_1, V_5] \times [P_1, P_4] = b_{20}$
- $b_{12} \cup b_{124} = [V_1, V_6] \times [P_1, P_4] = b_{24}$

La génération des blocs de données au travers de l'union permet également de définir l'inclusion entre blocs, puis de manière plus générale l'inclusion entre classes.

Définition 10. (*Inclusion d'intervalles*) Soient δ et δ' deux intervalles. On dit que δ est inclus dans δ' et on note $\delta \subseteq \delta'$ si $\min(\delta) \geq \min(\delta')$ et $\max(\delta) \leq \max(\delta')$.

Définition 11. (*Inclusion de blocs*) Soient $b = \delta_1 \times \dots \times \delta_k$ et $b' = \delta'_1 \times \dots \times \delta'_k$ deux blocs. On dit que b est inclus dans b' et on note $b \subseteq b'$ si $\forall \delta_i \in b, \delta'_i \in b', \delta_i \subseteq \delta'_i$

Par exemple, nous avons les inclusions de blocs suivantes :

- $b_{12} = [V_1, V_3] \times [P_1, P_4] \subseteq b_{16} = [V_1, V_4] \times [P_1, P_4]$
- $b_{123} = [V_3, V_5] \times [P_1, P_4] \subseteq b_{161} = [V_2, V_5] \times [P_1, P_4]$
- $b_{123} = [V_3, V_5] \times [P_1, P_4] \subseteq b_{162} = [V_3, V_5] \times [P_1, P_4]$

Nous définissons alors l'inclusion entre classes d'équivalences de la manière suivante :

Définition 12. (*Inclusion de classes d'équivalences*) Soient Cl et Cl' deux classes d'équivalences. On dit que Cl est inclus dans Cl' et on note $Cl \prec Cl'$ si $\{\forall b_i \in Cl, \exists b_j \in Cl' \mid b_i \subseteq b_j\}$

Afin d'illustrer notre propos, nous considérons l'exemple des classes d'équivalence Cl_{10} (tableau 6) et cl_{14} (tableau 7). Les inclusions de blocs suivantes montrent que $cl_{10} \prec Cl_{14}$:

| $Cl_{10} = \langle 3 \rangle \times \langle 2 \rangle$ | | |
|--------------------------------------------------------|------------------------------------------|------------------------------------------|
| $b_{100} = [P_1, P_3] \times [V_1, V_2]$ | $b_{104} = [P_1, P_3] \times [V_2, V_3]$ | $b_{108} = [P_1, P_3] \times [V_3, V_4]$ |
| $b_{101} = [P_2, P_4] \times [V_1, V_2]$ | $b_{105} = [P_2, P_4] \times [V_2, V_3]$ | $b_{109} = [P_2, P_4] \times [V_3, V_4]$ |
| $b_{102} = [P_3, P_5] \times [V_1, V_2]$ | $b_{106} = [P_3, P_5] \times [V_2, V_3]$ | $b_{110} = [P_3, P_5] \times [V_3, V_4]$ |
| $b_{103} = [P_4, P_6] \times [V_1, V_2]$ | $b_{107} = [P_4, P_6] \times [V_2, V_3]$ | $b_{111} = [P_4, P_6] \times [V_3, V_4]$ |

TAB 6 – Tous les blocs appartenant à la classe d'équivalence Cl_{10}

| $Cl_{14} = \langle 4 \rangle \times \langle 2 \rangle$ | | |
|--------------------------------------------------------|------------------------------------------|------------------------------------------|
| $b_{140} = [P_1, P_4] \times [V_1, V_2]$ | $b_{143} = [P_1, P_4] \times [V_2, V_3]$ | $b_{146} = [P_1, P_4] \times [V_3, V_4]$ |
| $b_{141} = [P_2, P_5] \times [V_1, V_2]$ | $b_{144} = [P_2, P_5] \times [V_2, V_3]$ | $b_{147} = [P_2, P_5] \times [V_3, V_4]$ |
| $b_{142} = [P_3, P_6] \times [V_1, V_2]$ | $b_{145} = [P_3, P_6] \times [V_2, V_3]$ | $b_{148} = [P_3, P_6] \times [V_3, V_4]$ |

TAB 7 – Tous les blocs appartenant à la classe d'équivalence Cl_{14}

- $b_{100} \subseteq b_{140}, b_{101} \subseteq b_{141}, b_{102} \subseteq b_{142}, b_{103} \subseteq b_{142}$
- $b_{104} \subseteq b_{143}, b_{105} \subseteq b_{144}, b_{106} \subseteq b_{145}, b_{107} \subseteq b_{145}$
- $b_{108} \subseteq b_{146}, b_{109} \subseteq b_{147}, b_{110} \subseteq b_{148}, b_{111} \subseteq b_{148}$

Notons que l'ensemble de ces classes muni de l'inclusion définie ci-dessus forme un treillis. La figure 3 montre le treillis d'inclusion des classes pouvant être générées à partir de l'exemple du tableau 3.

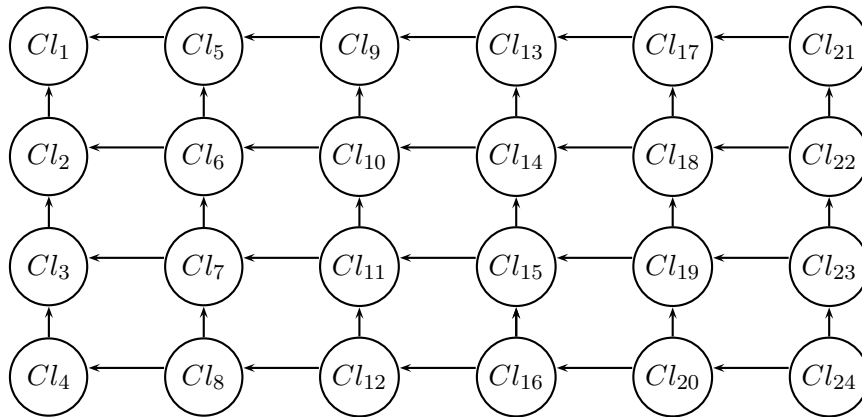


FIG 3 – Treillis d'inclusion des classes

Dans notre contexte, un tel treillis possède des propriétés intéressantes. Par exemple, avant de matérialiser en mémoire l'ensemble des blocs appartenant à une classe d'équivalence, il est possible de connaître leur fréquence (en multipliant les entiers la définissant). Cela permet alors de ne générer et tester que les blocs qui pourront a priori être fréquents. De plus, la génération d'une classe Cl' à partir d'une autre classe Cl se fait par une simple jointure des blocs appartenant à Cl . Dans la sous-section suivante, nous décrivons les algorithmes correspondants.

3.2 Algorithmes

L'algorithme 1 décrit la méthode principale d'extraction des blocs. A partir du treillis de la figure 3, nous définissons l'ordre de génération des blocs de données, en suivant l'ordre d'inclusion des classes (ligne 1). L'ensemble des classes dont la fréquence des blocs associée est de σ permettront d'atteindre toutes les classes de fréquence supérieure. Ces classes "racines" sont récupérées à la ligne 2. Elles définissent les blocs de plus petite taille. Ensuite, les classes suivantes sont générées récursivement en suivant l'ordre donné par le treillis. Cette étape est réalisée dans les lignes 3-5.

Algorithme 1 : ExtractBloc

Données : Un cube de données C ,
le seuil de fréquence minimal σ
le seuil de confiance minimal γ

Résultat : Tous les blocs respectant les seuils de fréquence et de confiance

```
1  $C \leftarrow \text{BuildLattice}(d_1, \text{null}, \text{null}, \text{null})$ 
2  $R \leftarrow \text{GetRoots}()$ 
3 pour chaque  $Cl \in C$  faire
4   |  $\text{RecursiveBloc}(Cl, \gamma)$ ;
5 fin
```

L'algorithme 2 réalise la construction du treillis des classes d'équivalence. Pour ce faire, l'algorithme s'exécute récursivement sur chaque intervalle de chaque dimension (lignes 9-14). Lorsqu'une classe d'équivalence est complètement construite et que sa fréquence respecte le seuil minimal, l'ensemble des classes directement incluses dans celle-ci sont calculées (lignes 3-6).

L'algorithme 3 permet de générer l'ensemble des blocs appartenant à une classe d'équivalence à partir d'une classe plus générale. Pour ce faire, certains blocs d'une classe sont unis deux à deux afin de générer l'ensemble des blocs de la classe marquée comme suivante par l'algorithme 2 (ligne 4-6). Cependant, l'algorithme n'effectue pas toutes les combinaisons de blocs d'une classe : en effet, seule une union sur les blocs ayant des intervalles consécutifs permettent d'atteindre la classe la plus proche dans le treillis (ligne 5). En reprenant l'exemple précédant, $b_{101} \cup b_{102} = b_{140} \subseteq Cl_{14}$, mais $b_{100} \cup b_{103} \notin Cl_{14}$.

Pour chaque nouveau bloc généré, la confiance peut être calculée en utilisant le principe d'inclusion-exclusion : soit b_i et b_j les deux blocs à joindre, m la valeur du bloc traitée et $x_{b_i \cap b_j}^m$ le nombre de cellules communes aux deux blocs à joindre (obtenues par intersection) contenant m . Alors¹

$$\text{Confiance}(b_i \cup b_j) = \text{Confiance}(b_i) + \text{Confiance}(b_j) - x_{b_i \cap b_j}^m$$

Une fois le treillis des inclusions construit, nous utilisons l'algorithme récursif 3 (en profondeur) générant toutes les classes supérieures et s'arrêtant avant l'obtention de la classe finale (tout le cube). Au fur et à mesure de la génération des classes de niveau k , les blocs de niveau $k - 1$ sont effacés de la mémoire, car ils ne seront plus réutilisés.

1. La confiance est exprimée ici en nombre de cellules et non en pourcentage

Algorithme 2 : BuildLattice

Données : la dimension courante d ,
la dimension précédente d_p ,
l'intervalle courant i ,
la classe Cl en cours de construction
le seuil de fréquence minimal σ

Résultat : Le treillis des classes d'équivalences

```
1 si  $d = D.last \wedge Freq(Cl) \geq \sigma$  alors
2    $C \leftarrow Cl$ 
3    $Cl.addSon(getClassEq(C, d, i - 1))$ 
4   pour chaque  $d' \in \{d - 1 \dots d_1\}$  faire
5      $Cl.addSon(getClassEq(C, d', i))$ 
6   fin
7   retourner  $C$ 
8 fin
9 sinon
10  pour chaque  $i \in \{0 \dots \mathcal{I}_d\}$  faire
11     $Cl = Cl \times \langle i \rangle$ 
12    BuildLattice ( $d + 1, d, i, Cl$ )
13  fin
14 fin
```

Algorithme 3 : RecursiveBloc

Données : Une classe d'équivalence Cl et tous les blocs lui appartenant
le seuil de confiance minimal γ

Résultat : Tous les blocs fréquents et respectant les seuils de fréquence et confiance

```
1  $Cl' \leftarrow \emptyset$ 
2 pour chaque  $d \in \mathcal{D}$  faire
3    $b' = null$ 
4   pour chaque  $b \in Cl$  faire
5     si  $IsConsecutive(b, b', d)$  alors
6        $Cl' \leftarrow b \cup b'$ 
7       pour chaque Mesure  $m$  faire
8          $conf = Conf(b) + Conf(b') - x_{b \cap b'}^m$ 
9         si  $conf \geq \gamma$  alors  $Output(b \cup b')$ 
10        fin
11      fin
12    fin
13     $RecursiveBloc(Cl')$ 
14     $Cl' \leftarrow \emptyset$ 
15 fin
```

3.3 Expérimentations

L'algorithme exposé ci-dessus a été testé en terme de performance en temps et en mémoire. Nous disposons d'un générateur de cube de données naïf, qui génère les mesures de manière aléatoire. Les cubes sont donc très denses : il n'y a pas de cellules vides. Ce type de cube ne reflète pas la réalité, mais permet néanmoins de tester le comportement de l'algorithme. Nous utilisons dans cette section trois jeux de données décrits par le tableau 8. Les expérimentations ont été menées sur un ordinateur Precision WorkStation R5400 Xeon(R) CPU E5450 3.00GHz doté de 16Go de RAM.

| Nom | $ \mathcal{D} $ | #membres / dim | #Cellules |
|--------|-----------------|----------------|-------------|
| D5V10 | 5 | 10 | 100000 |
| D5V100 | 5 | 100 | 10000000 |
| D7V10 | 7 | 10 | 10000000 |
| D10V10 | 10 | 10 | 10000000000 |

TAB 8 – Bases de données test pour l'extraction de blocs

Pour toutes ces expérimentations, nous fixons un seuil de confiance minimal très bas (10%), ce qui augmente nos chances d'extraire des blocs. Le temps d'extraction dépend très fortement du nombre de dimensions ainsi que du nombre de membres par dimension. Les expérimentations montrent que le temps d'extraction peut être très long et ne permet pas d'extraction en temps réel. En revanche, on note que l'exploitation en terme de mémoire n'est pas très élevée et reste constante du moment où les valeurs du cube sont chargées en mémoire. Les figures 4a et 4b montre les temps d'extractions nécessaires pour une valeur de fréquence minimale variant de 90% à 40%. On note qu'avec 5 dimensions, notre algorithme extrait en moins de 600 secondes un peu plus de 100000 blocs de données. En revanche, il faut plus de 1400 minutes pour extraire environ 40000 blocs avec 7 dimensions. Pour toutes ces expérimentations, la charge en mémoire n'a pas excédé 700Mo.

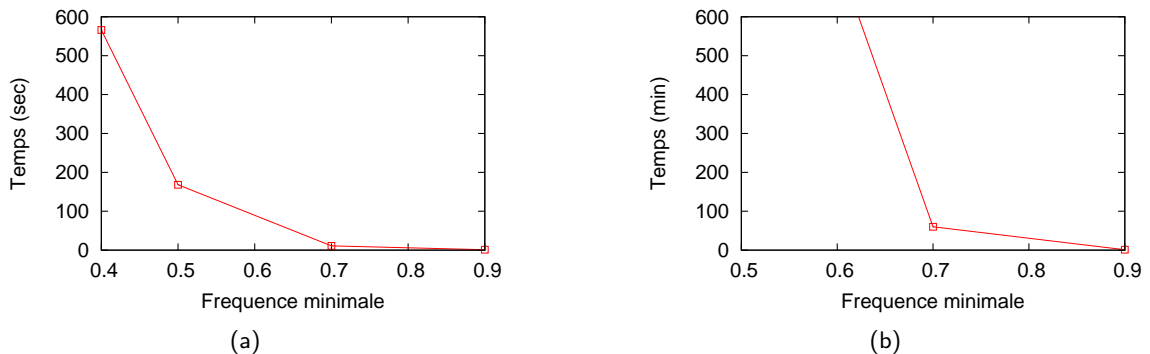


FIG 4 – Temps d'exécution pour (a) D5V10 (b) D7V10, (c) D10V10

Lors de l'extraction de blocs de données, le seuil de fréquence fixe par avance le nombre de blocs différents qu'il faudra explorer. C'est le seuil de confiance minimal qui permet ou non d'afficher un bloc en résultat. C'est ce que montre la figure 6 : le temps d'extraction varie très peu pour un même seuil de fréquence minimal et des seuils de confiance différents.

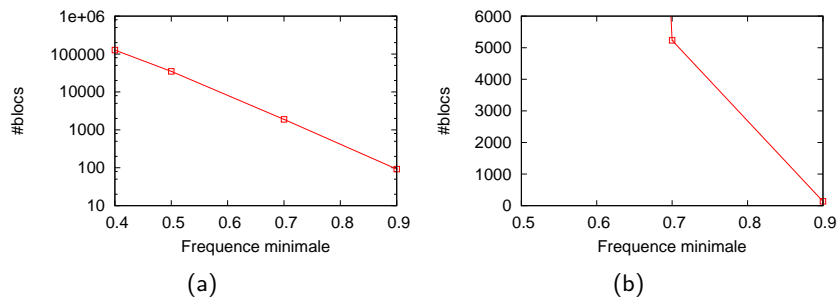


FIG 5 – Temps d'exécution pour (a) D5V10 (b) D7V10

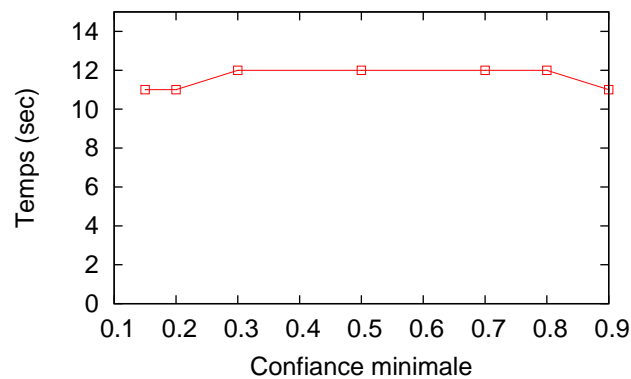


FIG 6 – Variation du seuil de confiance pour D5V10, avec minFreq = 40%

Optimisations

La méthode présentée ci-dessus est perfectible au travers de diverses optimisations. Tout d'abord, les expérimentations menées ci-dessus montrent que les performances de l'algorithme dépendent fortement du nombre de membres par dimension ainsi que du seuil de confiance minimal. Or, notre algorithme procède par partitionnement du cube en sous-cubes constituant des blocs de données. La jointure entre ces sous-cubes se réalise facilement, d'autant plus lorsque les intervalles des membres d'une dimension sont contigus. Ces éléments montrent que les performances en terme de temps de notre méthode peuvent être grandement améliorées par des techniques de parallélisation, qui peuvent aisément être mises en œuvre.

Actuellement, notre méthode ne tient pas compte des cellules vides. Pourtant, dans les jeux de données réels, ces cellules constituent la majorité du cube. Il conviendrait donc de les prendre en compte de manière plus efficace, notamment au travers d'un prétraitement. Par exemple, ces ensembles de cellules peuvent elles-mêmes constituer des blocs de données, qu'il suffirait de prendre en compte lors de la génération des blocs. Le gain en terme de temps et mémoire serait alors non négligeable.

Dans la section suivante, nous présentons comment s'appuyer sur ces blocs pour extraire des motifs graduels multidimensionnels.

4 Extraction de règles graduelles multidimensionnelles

Dans cette section, nous expliquons comment extraire des itemsets graduels multidimensionnels à partir de blocs de données.

4.1 Les DG-sets

Nous souhaitons extraire des corrélations de variation de mesures (valeurs des blocs) associées à des dimensions dont les membres sont ordonnés. Par exemple, à partir du cube du tableau 2a, nous souhaiterions extraire “*Plus le nombre de jours de maladie diminue et plus le stade de la maladie augmente, alors plus la valeur des blocs augmente*”.

De manière plus générique, on peut voir cette gradualité comme “*Plus (moins) d_1, \dots , et plus (moins) d_n , alors plus la valeur des blocs augmente (diminue)*”. Ce type de corrélations, que nous appellerons **DCG**, est clairement composée de corrélations de variations sur deux ensembles distincts :

- Les dimensions en elles-mêmes (première partie de la règle)
- La mesure (seconde partie de la règle)

La première partie concerne la gradualité sur les dimensions, ce qui revient à comparer les membres des dimensions. Ainsi, dans notre approche, nous considérons que le cube de données contient des dimensions ordonnées :

Définition 13. (*Dimension ordonnée*) Une dimension d est ordonnée si son domaine est muni d'une relation d'ordre total.

Par exemple, les dimensions d_1 et d_2 du cube C sont ordonnées, car elles peuvent être munie d'une relation d'ordre total : nous avons, pour d_1 : $10 \leq 20 \leq 30 \leq 40 \leq 50$, et pour d_2 : $1 \leq 2 \leq 3 \leq 4 \leq 5$. La définition 13 nous permet d'introduire la notion de gradualité sur les dimensions. Ainsi, nous avons les notions sémantiques “*le nombre de jours augmente*” ou “*le nombre de jours diminue*”.

Dans ce chapitre, nous ne considérons que les dimensions ordonnées, ce qui signifie que les dimensions qui ne sont pas ordonnables seront ignorées par notre méthode.

La seconde partie concerne l'augmentation ou la diminution de la mesure, au travers de l'utilisation des blocs. De manière plus formelle, nous définissons une DCG de la manière suivante :

Définition 14. (*Dimension graduelle*) Une 1-DG est de la forme $[d^*, *m]$, où d^* est une dimension graduelle telle que $*m \in \{\leq, \geq\}$ est un opérateur se rapportant à la mesure (valeur des blocs).

Notons l'utilisation des opérateurs de comparaison $\{\leq, \geq\}$, qui permettent de conserver les cubes ayant des valeurs égales. Cela nous permet de maximiser l'ensemble des règles supportant une règles. Toutefois, les blocs de valeurs égales participeront à la fois au support de l'augmentation et de la diminution.

Définition 15. (*DG-set*) Soit $C = \langle dom_1, \dots, dom_k, dom_m, m_C \rangle$ un cube. Une DG-set est de la forme $[\{d_l^{*l}, \dots, d_i^{*i}\}, *m]$, où $\{d_l^{*l}, \dots, d_i^{*i}\}$ est un ensemble de dimensions graduelles telles que $\forall j = 1..i, d_j \in C$ et $*m \in \{\leq, \geq\}$ est un opérateur se rapportant à la mesure (valeur des blocs).

Dans cet article, nous considérons que les règles multidimensionnelles graduelles peuvent être générées à partir des DG-Set en post-traitement. Par abus de langage, une règle multidimensionnelle graduelle est en réalité un DG-Set. Dans notre contexte, comparer deux mesures revient à comparer deux blocs. Nous définissons donc l'ordre entre blocs de la manière suivante :

Définition 16. (ordre entre blocs) Soit $b = \delta_1 \times \dots \times \delta_n$ et $b' = \delta'_1 \times \dots \times \delta'_n$ deux blocs définis sur les dimensions d_1, \dots, d_n ayant pour valeur associée m et m' respectivement. Soit $r = [\{d_1^{*1}, \dots, d_n^{*n}\}, *m]$ une DG-set. On dit que b précède b' en fonction de r si :

- $m *_{r_1} m'$
- $\forall j \in \{1, \dots, k\}, \min(\delta_j) *_{r_1} \min(\delta'_j) \wedge \max(\delta_j) *_{r_1} \max(\delta'_j)$

On note $b \triangleleft_{r_1} b'$.

Par exemple, lorsque l'on considère la règle multidimensionnelle graduelle $r_1 = [\{CSP^{\leq}\}, \geq]$ (Plus le stade de la maladie diminue, plus la valeur des blocs augmente), nous avons $b_1 \leq b_2$. De plus, $\min([1, 2]) \leq \min([2, 4]) \wedge \max([1, 2]) \leq \max([2, 4])$ b_1 précède donc b_2 ($b_1 \triangleleft_{r_1} b_2$). En revanche, si l'on considère $b_1 = [1]$ et $b_4 = [1, 2]$, nous n'avons ni $b_1 \triangleleft_{r_1} b_4$, ni $b_4 \triangleleft_{r_1} b_1$, car $\min([1]) = \min([1, 2]) \wedge \max([1]) \leq \max([1, 2])$.

La généralisation à n blocs ordonnés se fait alors de la manière suivante :

Définition 17. (Liste de blocs ordonnés) Soit $r = [\{d_1^{*1}, \dots, d_n^{*n}\}, *m]$ une DG-set. Soit x un entier appartenant à \mathbb{N}^* . Une liste de taille x de blocs $\mathcal{L} = \langle b_1, \dots, b_x \rangle$ respecte r si $\forall i, j \in \{1, \dots, x\} b_i \triangleleft_r b_j$.

Par exemple, deux listes respectent la règle $[\{CSP^{\leq}\}, \geq]$: $\mathcal{L}_1 = \langle b_3, b_2, b_4 \rangle$ et $\mathcal{L}_2 = \langle b_1, b_2, b_4 \rangle$. Afin de mesurer la représentativité d'une règle sur un cube, nous proposons d'utiliser une mesure de fréquence définie de la manière suivante :

Définition 18. Soit C un cube, \mathcal{B} le nombre de blocs extraits sur ce cube et $\mathcal{G}_r = \{\mathcal{L}_1 \dots \mathcal{L}_z\}$ l'ensemble de toutes les listes respectant r . Alors $Freq(r) = \frac{\max_{1 \leq i \leq z} (|\mathcal{L}_i|)}{\mathcal{B}}$

4.2 Propriétés des DG-sets

Dans cette partie, nous montrons que nos définitions sont compatibles avec les propriétés classiques en fouille de données. Ainsi, nous retrouvons par exemple la propriété d'anti-monotonie. Pour ce faire, nous redéfinissons la notion d'inclusion de la manière suivante :

Définition 19. (Inclusion) Soient $r = [\{d_1^{*1}, \dots, d_n^{*n}\}, *m]$ et $r' = [\{d_1^{*1'}, \dots, d_n^{*n'}\}, *m']$ deux DG-sets. r est inclus dans r' si

- $*m = *m'$
- $\forall d (d \in \{d_1^{*1}, \dots, d_n^{*n}\} \Rightarrow d \in \{d_1^{*1'}, \dots, d_n^{*n'}\})$

On note $r \sqsubseteq r'$

Par exemple, $[\{d_1^{\leq}, d_2^{\geq}\}, \leq] \sqsubseteq [\{d_1^{\leq}, d_2^{\geq}, d_3^{\geq}\}, \leq]$. Par contre, $[\{d_1^{\leq}, d_2^{\geq}\}, \leq]$ n'est pas inclus dans $[\{d_1^{\leq}, d_2^{\geq}, d_3^{\geq}\}, \geq]$.

Proposition 1. (Anti-monotonie DG-set) Soient r et r' deux DG-sets, nous avons : $r \sqsubseteq r' \Rightarrow Freq(r) \geq Freq(r')$.

Démonstration. Soient deux DG-set r_k et r_{k+1} tels que $r_k \subseteq r_{k+1}$, avec k et $k+1$ la longueur de ces DG-sets. Soit ml la liste de taille maximale \mathcal{G}_{r_k} . Nous avons $\forall b, b' \in ml$:

- si $\neg(b \triangleleft_{r_{k+1}} b')$ alors b' ne fera pas partie de \mathcal{G}_{r_k} et par conséquent $Freq(r_k) > Freq(r_{k+1})$
- si $(b \triangleleft_{r_{k+1}} b')$, alors b' fera partie de \mathcal{G}_{r_k} et par conséquent $Freq(r_k) = Freq(r_{k+1})$

Ainsi, nous avons $Freq(r_k) \geq Freq(r_{k+1})$. □

Comme dans les méthodes d'extraction de connaissance classiques, l'anti-monotonie des DG-sets permet de tronquer l'espace de recherche dès qu'un ensemble ne respecte pas le support minimal. Cependant, le nombre de combinaisons différentes de corrélations graduelles à considérer reste plus élevé que pour l'extraction d'itemsets. Par exemple, pour n dimensions, il existe 2^{n+1} DG-sets à tester (contre 2^n dans le cas classique). Comme dans le chapitre ??, nous utilisons la notion de complémentarité due à la gradualité afin de réduire l'espace de recherche. De même que pour les motifs graduels présentés précédemment nous avons :

Définition 20. (complémentaire) Soit $r = [\{d_1^{*1}, \dots, d_n^{*n}\}, *'_m]$ une DG-set. Sa DG-set complémentaire est $c(r) = [\{d_1'^{*1}, \dots, d_n'^{*n}\}, *'_m]$ si $\forall j \in [1, n] d_j = d'_j$ et $*'_j = c_*(^*'_j)$ et $*'_m = c_*(^*'_m)$, où $c_*(\geq) = \leq$ et $c_*(\leq) = \geq$.

Par exemple, $c([\{d_2^{\geq}\}, \leq]) = [\{d_2^{\leq}\}, \geq]$, mais $c([\{d_2^{\geq}\}, \leq]) \neq [\{d_2^{\geq}\}, \geq]$.

Proposition 2. Soit r un DG-set tel que $c(r)$ est le complémentaire de r . Alors l'ensemble des listes composant \mathcal{G}_r est le même que celles composant $\mathcal{G}_{c(r)}$.

Corollaire 1. $Freq(r) = Freq(c(r))$

Le corollaire 1 montre que le support de la moitié des DG-sets peut être déduit de manière automatique. Il ne sera donc pas nécessaire de les générer. D'autre part, il se trouve que ces DG-sets sont en réalité des informations redondantes. En effet, ce corollaire montre que "Plus le nombre de jours de maladie diminue et plus le stade de la maladie augmente alors plus la valeur augmente" est exactement la même chose que "Moins le nombre de jours de maladie diminue et moins le stade de la maladie augmente alors plus la valeur diminue".

4.3 Algorithme

La méthode adoptée afin d'extraire les règles graduelles multidimensionnelles repose sur les algorithmes décrits au chapitre ??. Cependant, dans les chapitre précédents, l'espace de recherche était composé des attributs de la base, et la fréquence reposait sur le nombre d'objets de la base dont les variations de valeurs suivaient le sens de l'itemset graduel considéré. Dans le contexte multidimensionnel, et plus particulièrement celui basé sur les blocs, l'espace de recherche et les objets sont différents. Ici, l'espace de recherche est constitué des dimensions. La fréquence est basée sur le nombre de blocs.

Nous utilisons un algorithme par niveau qui augmente à chaque passe le nombre de dimensions graduelles. Ainsi, à l'image de l'algorithme Apriori de [AS94], nous construisons un arbre des préfixes. Dans cette structure, chaque nœud contient une dimension graduelle associée à un opérateur graduel sur la mesure (correspondant à la seconde partie de la règle). Le chemin d'un nœud à la racine représente une DG. Le corollaire 1 nous permettant de ne générer que la moitié des DG, nous illustrons dans ce chapitre l'extraction de connaissances graduelles sur l'augmentation de la mesure (les supports des diminutions

sont obtenues en inversant les opérateurs). De plus, nous rappelons que les dimensions considérées sont des dimensions ordonnées, les dimensions non ordonnées étant ignorées.

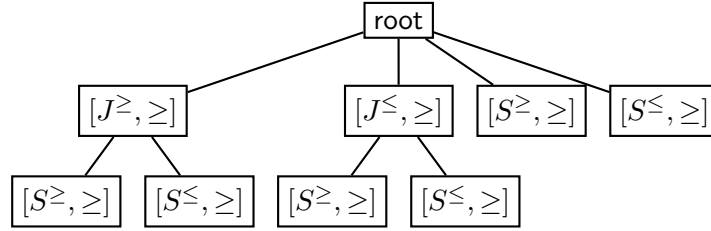


FIG 7 – Un exemple d'arbre préfixé pour l'extraction de DG-sets

La figure 7 montre l'arbre des préfixes généré pour extraire des DG-sets à partir du tableau 2a. L'arbre s'étend sur deux niveaux car il n'y a que deux dimensions. Nous remarquons que nous avons généré 4 noeuds au niveau 2, au lieu des 8 recouvrant la totalité des corrélations graduelles possibles. D'autre part, l'arbre généré est déséquilibré, ce qui permet d'éviter toute redondance.

Apriori est un algorithme *par niveau*, c'est-à-dire que que les noeuds de niveau k sont générés par jointure sur les noeuds du niveau $k - 1$. L'algorithme exploite ainsi la propriété d'anti-monotonie en alternant les étapes de génération et de comptage jusqu'à ne plus avoir de candidats fréquents.

L'extraction de gradualité nécessite de conserver pour chaque candidat généré l'ensemble des ordres possibles (l'ensemble \mathcal{G}_r), afin de considérer le meilleur lors de la génération du DG-set de taille $k + 1$. Cependant, cet ensemble dépend fortement du nombre de blocs extraits. Ainsi, nous utilisons la structure binaire proposée précédemment. Cette matrice binaire est définie sur les blocs, et non plus sur les objets de la base :

$$\forall a, b \in \{1, \dots, x\} \times \{1, \dots, x\}, t_a, t_b \in \mathcal{T}_{\mathcal{G}_r} \begin{cases} m_{t_a, t_b} = 1 & \text{si } t_a \triangleleft_r t_b, \\ m_{t_a, t_b} = 0 & \text{sinon} \end{cases}$$

En reprenant l'exemple précédent, l'algorithme construit lors de la première passe les matrices représentées par les tableaux 9a et 9b :

| \uparrow | b_1 | b_2 | b_3 | b_4 |
|------------|-------|-------|-------|-------|
| b_1 | 1 | 1 | 0 | 1 |
| b_2 | 0 | 1 | 0 | 1 |
| b_3 | 0 | 1 | 1 | 1 |
| b_4 | 0 | 0 | 0 | 1 |

(a)

| \uparrow | b_1 | b_2 | b_3 | b_4 |
|------------|-------|-------|-------|-------|
| b_1 | 1 | 1 | 0 | 0 |
| b_2 | 0 | 1 | 0 | 0 |
| b_3 | 0 | 0 | 1 | 0 |
| b_4 | 0 | 1 | 0 | 1 |

(b)

TAB 9 – Matrice binaire pour (a) $[J^{\leq}, \geq]$ et (b) $[S^{\geq}, \geq]$

Ensuite, l'algorithme de calcul de fréquence glouton présenté au chapitre ?? est utilisé. A l'issu de ce processus, nous avons extrait en deux temps des règles graduelles multidimensionnelles basées sur les blocs.

5 Expérimentations

Dans cette section, nous décrivons les expérimentations menées. Nous avons implémenté les algorithmes présentés ci-dessus en C++. Ces algorithmes ont été testés en terme de temps, de mémoire et de nombre de motifs extraits. Pour ce faire, les blocs utilisés sont générés de manière aléatoire, en prenant en compte le nombre de dimensions ($|D|$), le nombre de membres par dimension (compris entre 0 et 10), le nombre de valeurs de blocs différentes ($|V|$) ainsi que le nombre de blocs ($|B|$). Nous avons généré les trois fichiers présentés dans le tableau 10 :

| Name | $ D $ | $ V $ | $ B $ |
|--------------|-------|-------|-------|
| D5V10B40 | 5 | 10 | 40 |
| D10V100B500 | 10 | 100 | 500 |
| D10V100B5000 | 10 | 100 | 5000 |

TAB 10 – Spécifications des jeux de test

Les expérimentations ont été menées afin de mesurer les consommations en terme de temps, et mémoire en fonction du support minimal. De plus, nous avons conservé le nombre de DG-sets extraits. Les jeux de données étant générés de manière aléatoire, il est nécessaire de baisser le support afin de trouver des DG-sets. Les expérimentations ont été menées sur un serveur possédant un processeur Intel(R) Xeon(R) CPU E5450 @ 3.00GHz, et ayant 16Go de mémoire vive.

Les résultats obtenus sont satisfaisants en terme de temps d'exécution et de mémoire. Ainsi, pour un jeu contenant un faible nombre de dimensions et de blocs, il faut environ 1 seconde afin d'extraire environ 250 DG-sets. L'algorithme est particulièrement sensible au nombre de blocs, plus qu'au nombre de dimensions et de valeurs de dimensions. C'est ce que montrent les figures 8a, 9a et 10c. En effet, le nombre de blocs, fixé à 5000 dans le jeu de données D10V100B500, rend le temps d'exécution plus long : environ 17 minutes pour un support minimal fixé à 0.1 sont nécessaires à l'extraction de 80 DG-sets. En revanche, pour le même nombre de dimensions et le même support, mais seulement 500 blocs, il faut environ 30 secondes afin d'extraire 120 DG-sets.

Nous avons également exécuté notre algorithme sur le jeu de données reflétant la réalité *Chess Endgame Database for White King and Rook against Black King (KRK) – Black-to-move Positions Drawn or Lost in N Moves*². Ce jeu calcule, pour une position donnée d'un roi blanc, d'une tour blanche et d'un roi noir, le nombre de coups à jouer de manière optimale afin de mener à la victoire de la partie blanche. Notons que pour certaines positions, il peut y avoir match nul (considéré comme valeur nulle dans notre cas). Le cube de données a alors été construit de la manière suivante :

- la dimension 1 ($D1$) représente la distance euclidienne entre le roi blanc et le roi noir,
- la dimension 2 ($D2$) représente la distance euclidienne entre le roi blanc et la tour blanche,
- la dimension 3 ($D3$) représente la distance euclidienne entre le roi noir et la tour blanche.

2. [http://archive.ics.uci.edu/ml/datasets/Chess+\(King-Rook+vs.+King\)](http://archive.ics.uci.edu/ml/datasets/Chess+(King-Rook+vs.+King))

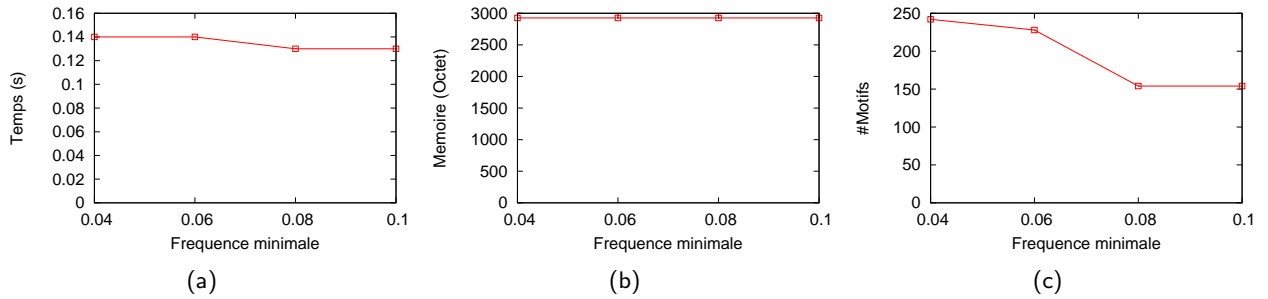


FIG 8 – Pour D5V10B40, en fonction du support (a) Temps d'exécution, (b) Mémoire utilisée, (c) Nombre de DG-sets extraits

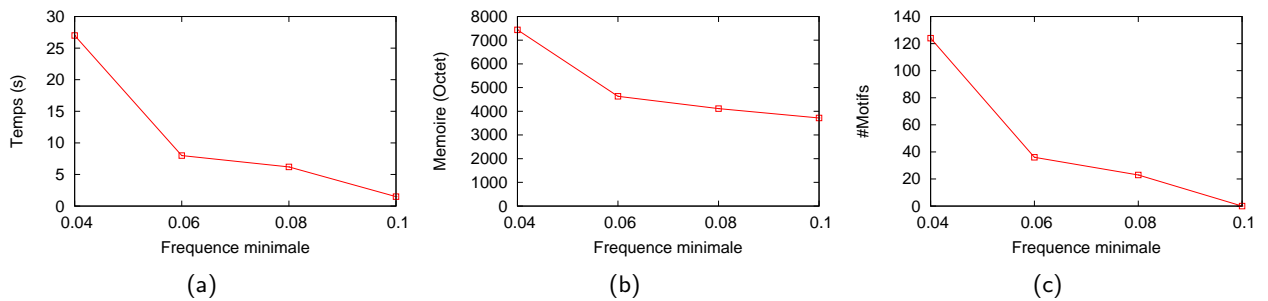


FIG 9 – Pour D10V100B500, en fonction du support (a) Temps d'exécution, (b) Mémoire utilisée, (c) Nombre de DG-sets extraits

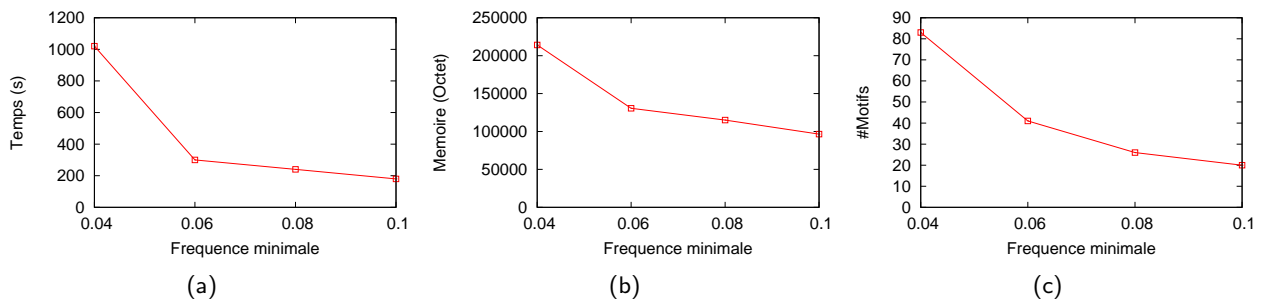


FIG 10 – Pour D10V100B5000, en fonction du support (a) Temps d'exécution, (b) Mémoire utilisée, (c) Nombre de DG-sets extraits

Le jeu contient 28056 instances réparties sur 648 cellules après calcul du cube en agrégeant sur les attributs non présents dans le schéma multidimensionnel (group by). La mesure d'une cellule contient le nombre de pas à jouer afin de mener à la victoire des blancs, et null si match nul. L'extraction de blocs a été effectuée avec un support de 5 cellules, et une confiance minimale de 80% (4 cellules sur 5). Nous avons obtenu 1816 blocs différents. Nous avons ensuite extrait les blocs multidimensionnels graduels en utilisant un faible support : 2%. Le tableau 11 résume les résultats obtenus.

| | | | |
|---------------------------|---------------------------|----------------------------------|----------------------------------|
| $D1 \geq D2 \geq$, 18.1% | $D1 \leq D2 \geq$, 10.9% | $D1 \geq D2 \geq D3 \geq$, 4.6% | $D1 \leq D2 \geq D3 \geq$, 2.5% |
| $D1 \geq D2 \leq$, 10.2% | $D1 \leq D2 \leq$, 10.2% | $D1 \geq D2 \geq D3 \leq$, 4% | $D1 \leq D2 \geq D3 \leq$, 3.5% |
| $D1 \geq D3 \geq$, 12.7% | $D1 \leq D3 \geq$, 12.1% | $D1 \geq D2 \leq D3 \geq$, 2.6% | $D1 \leq D2 \leq D3 \geq$, 3.6% |
| $D1 \geq D3 \leq$, 14.1% | $D1 \leq D2 \leq$, 18.7% | $D1 \geq D2 \leq D3 \leq$, 2.7% | $D1 \leq D2 \leq D3 \leq$, 5% |
| $D2 \geq D3 \geq$, 12.6% | $D2 \leq D3 \geq$, 9.3% | | |
| $D2 \geq D3 \leq$, 17.1% | $D2 \leq D3 \leq$, 14.5% | | |

TAB 11 – Blocs graduels obtenus sur le jeu de données Chess

De ces expérimentations, nous déduisons les règles suivantes : plus la distance entre le roi blanc et le roi noir est élevée et plus la distance entre le roi noir et la tour blanche est élevée alors moins le nombre de coups à jouer pour gagner diminue. En revanche, si l'on ajoute que la distance entre la tour blanche et le roi blanc augmente (respectivement diminue) alors le support du nombre de coups à jouer passe à 5% (respectivement 3.5%). De manière générale, nous déduisons de ces résultats que les manière de gagner optimales se jouent entre deux distances : soit la distance entre le roi blanc et la tour blanche est faible, soit la distance entre le roi noir et la tour blanche est élevée. En revanche, nous montrons que ces trois distances ne sont pas liées par une co-variation graduelle puisqu'aucun résultat n'est produit par notre algorithme.

6 Discussion

Dans ce chapitre, nous proposons une approche originale permettant d'extraire, à partir de cubes de données, des règles graduelles multidimensionnelles de la forme *Plus le nombre de jours de maladie augmente et plus le stade de la maladie augmente, plus le nombre de patients est grand*. Ces règles sont extraites en considérant des dimensions ordonnées et des blocs de données extraits selon la valeur de la dimension. Cette approche permet de dégager les tendances qui sont présentes dans les cubes de données. Nous nous appuyons pour ce faire sur une méthode de découverte à partir d'algorithmes par niveau en faisant croître le nombre de dimensions présentes dans les règles graduelles générées, et en considérant une représentation binaire pour représenter les ordres entre blocs décrivant la valeur de mesure en fonction des valeurs des dimensions. Nos expérimentations prouvent que l'algorithme est efficace en terme d'utilisation mémoire, et qu'il est fortement dépendant du nombre de blocs. Le nombre de dimensions en revanche influe peu sur le temps d'exécution.

La sémantique des règles extraites peut-être améliorée, notamment en intégrant la notion de hiérarchie lors de l'extraction. En effet, l'une des particularités des cubes de données est d'associer à chaque

dimension une hiérarchie permettant ainsi à l'utilisateur d'affiner le niveau de granularité des informations lors de sa navigation. Cependant, l'intégration de ces structures de données au processus de fouille peut s'avérer compliqué notamment lors de l'agrégation au niveau hiérarchique supérieur, comme le montrent [IKA02, DHL⁺04] ou encore [PLT08]. Dans notre contexte, la mesure choisie lors de l'agrégation conditionne la dernière partie de la règle graduelle multidimensionnelle, puisqu'elle influera directement sur les valeurs associées aux blocs extraits. Toutefois, l'intégration de règles graduées généralisées permettrait d'améliorer le support et donc la pertinence des résultats présentés à l'utilisateur.

Dans [CLL07], les auteurs utilisent la confiance comme contrainte d'élagage des blocs. Bien que non-antimonotone, cela permet d'explorer des blocs qui n'auraient jamais pu être atteints en utilisant uniquement la contrainte de fréquence. Cela demande une étude théorique préalable qui permettrait de borner les erreurs lors des générations suivantes.

Enfin, notre méthode ne prend pas en compte le recouvrement des blocs. En cas de recouvrement total, nous prenons en compte le bloc de taille maximale afin de présenter la règle la plus représentative à l'utilisateur. Cependant, ces recouvrements peuvent apporter potentiellement des informations intéressantes, puisque plus ciblées. D'autre part, les "chevauchements", sans recouvrement total, offrent une sémantique différente, que nous n'avons pas exploitée au travers de la méthode proposée.

En conclusion, l'extraction de motifs gradués à partir de bases multidimensionnelles est prometteuse, notamment dans le contexte actuel de construction de nombreux entrepôts de données médicaux. Il s'agira alors de valider expérimentalement les approches décrites dans ce chapitre sur de gros volumes de données. Au sein de ces entrepôts de données, la prise en compte de l'aspect historisé des données sera importante.

Dans le chapitre suivant, nous abordons l'impact de cette prise en compte pour découvrir des motifs gradués intégrant la notion de temporalité. Nous étudions pour ce faire les liens entre motifs gradués et bases de données séquentielles. Nous nous intéressons également aux liens entre les motifs gradués, Pareto et les skylines.

Références

- [AS94] Rakesh AGRAWAL et Ramakrishnan SRIKANT : Fast Algorithms for Mining Association Rules. *In 20th International Conference on Very Large Data Bases, (VLDB'94)*, pages 487–499, 1994.
- [CLL07] Yeow Wei CHOONG, Anne LAURENT et Dominique LAURENT : Summarizing data cubes using blocks. *In P. Poncelet M. Teisseire F. MASSEGLIA, éditeur : Data Mining Patterns : New Methods and Applications*, page 36. IDEA Group Inc., 2007.
- [CLL08] Yeow Wei CHOONG, Anne LAURENT et Dominique LAURENT : Mining multiple-level fuzzy blocks from multidimensional data. *Fuzzy Sets and Systems*, 159(12), 2008.
- [CLM03] Yeow Wei CHOONG, Dominique LAURENT et Patrick MARCEL : Computing appropriate representations for multidimensional data. *Data Knowl. Eng.*, 45(2):181–203, 2003.

- [CMLL04] Yeow Wei CHOONG, Pierre MAUSSION, Anne LAURENT et Dominique LAURENT : Summarizing multidimensional databases using fuzzy rules. *In Proc. of the 10th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IP-MU'04)*, pages 99–106, 2004.
- [DHL⁺04] Guozhu DONG, Jiawei HAN, Joyce LAM, Jian PEI, Ke WANG et Wei ZOU : W. : Mining constrained gradients in large databases. *IEEE Transactions on Knowledge Discovery and Data Engineering*, 16, 2004.
- [IKA02] Tomasz IMIELINSKI, Leonid KHACHIYAN et Amin ABDULGHANI : Cubegrades : Generalizing association rules. *Data Mining and Knowledge Discovery*, 6:219–258, 2002.
- [KHC97] Micheline KAMBER, Jiawei HAN et Jenny Y. CHIANG : Metarule-guided mining of multidimensional association rules. *In Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, 1997.
- [Lau02] Anne LAURENT : *Bases de données multidimensionnelles floues et leur utilisation pour la fouille de données*. Thèse de doctorat, université Paris 6, septembre 2002.
- [PCL⁺05] Marc PLANTEVIT, Yeow Wei CHOONG, Anne LAURENT, Dominique LAURENT et Maguelonne TEISSEIRE : M2SP : Mining Sequential Patterns Among Several Dimensions. *In 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05)*, Lecture Notes in Computer Science, pages 205–216. Springer, octobre 2005.
- [PGG⁺07] Marc PLANTEVIT, Sabine GOUTIER, Françoise GUISEL, Anne LAURENT et Maguelonne TEISSEIRE : Mining unexpected multidimensional rules. *In ACM tenth international workshop on Data warehousing and OLAP (DOLAP'07)*, pages 89–96, 2007.
- [PHP⁺01] Helen PINTO, Jiawei HAN, Jian PEI, Ke WANG, Qiming CHEN et Umeshwar DAYAL : Multidimensional sequential pattern mining. *In CIKM '01 : Proceedings of the tenth international conference on Information and knowledge management*, pages 81–88, New York, NY, USA, 2001. ACM.
- [PJ98] Torben Bach PEDERSEN et Christian S. JENSEN : Research issues in clinical data warehousing. *In SSDBM '98 : Proceedings of the 10th International Conference on Scientific and Statistical Database Management*, pages 43–52, Washington, DC, USA, 1998. IEEE Computer Society.
- [PL08] Sellappan PALANIAPPAN et Chua Sook LING : Clinical decision support using olap with data mining. *International Journal of Computer Science and Network Security*, septembre 2008.
- [PLT07] Marc PLANTEVIT, Anne LAURENT et Maguelonne TEISSEIRE : Extraction d'outliers dans des cube de données : une aide à la navigation. *In Revue des Nouvelles Technologies DE L'INFORMATION*, éditeur : *Entrepôts de Données et Analyse en ligne (EDA'07)*, pages 113–130, Poitiers, 2007.
- [PLT08] Marc PLANTEVIT, Anne LAURENT et Maguelonne TEISSEIRE : Up and Down : Mining Multidimensional Sequential Patterns Using Hierarchies. *In Johann Eder Tho Manh Nguyen IL-YEOL SONG*, éditeur : *Data Warehousing and Knowledge Discovery, 10th International Conference, DaWaK 2008*, Lecture Notes in Computer Science, pages 156–165. Springer Berlin / Heidelberg, septembre 2008.