Clustering sous contraintes utilisant SAT

Jean-Philippe Metivier, Patrice Boizumault, Bruno Crémilleux, Medhi Khiari, Loudni Samir

Clustering sous contraintes utilisant SAT

J.-P. Métivier P. Boizumault B. Crémilleux M. Khiari S. Loudni

Université de Caen – GREYC – (CNRS UMR 6072)
Campus II – Côte de Nacre – Boulevard du Maréchal Juin
14000 Caen – France
prenom.nom@unicaen.fr

Résumé

Le clustering a pour objectif de partitionner un ensemble de données (transactions) en groupes (clusters) d'une manière telle que les transactions appartenant à un même cluster soient similaires mais différentes de celles appartenant aux autres clusters. Le clustering sous contraintes a pour objectif de trouver des clusters plus pertinents en spécifiant, sous forme de contraintes, les propriétés requises. Nous avons proposé [16] un langage à base de contraintes pour l'extraction de motifs combinant plusieurs motifs locaux. Dans cet article, nous nous intéressons à la mise en œuvre en SAT de notre langage de contraintes. Pour cela, nous présentons le codage réalisé et nous montrons comment il tire parti des caractéristiques des solveurs SAT. Les expérimentations effectuées avec `MiniSat` montrent la faisabilité et l'intérêt de notre approche.

Abstract

Clustering aims at partitioning data into groups (clusters) so that transactions occurring in the same cluster are similar but different from those appearing in other clusters. Clustering is an important and popular data mining task and, by nature, clustering proceeds by iteratively refining queries until a satisfactory solution is found. In this application paper, we first present a constraint based language for modeling such queries. The usefulness of our approach is highlighted by several examples coming from the clustering based on associations. Then we show how each constraint (and each query) can be encoded in SAT and solved taking benefit from several features of SAT solvers. Experiments performed using `MiniSat` show the feasibility and the interest of our approach.

1 Introduction

Le clustering est une activité importante en fouille de données dont l'objectif est de partitionner un ensemble de données (transactions) en groupes (clusters) d'une manière telle que les transactions appartenant à un même cluster soient similaires mais différentes de celles appartenant aux autres clusters [12].

Le clustering sous contraintes [3] a pour objectif de trouver des clusters plus pertinents en spécifiant, sous forme de contraintes, les propriétés requises. Parmi les contraintes les plus communément utilisées figurent les contraintes imposant que des transactions appartiennent (ou non) à un même cluster [20], que les clusters aient une taille minimale et/ou maximale [2], qu'une transaction appartienne à un cluster donné, ...

Dans un travail précédent [16], nous avons proposé un langage à base de contraintes pour l'extraction de motifs combinant plusieurs motifs locaux. L'utilisateur modélise son problème sous forme d'une requête initiale (conjonction de contraintes), puis procède par raffinements successifs de requêtes jusqu'à ce qu'une solution satisfaisante soit trouvée. Le langage de contraintes proposé porte sur des termes construits à partir de constantes, de variables, d'opérateurs et de symboles de fonctions. Cette approche tire bénéfice des travaux récents sur la fertilisation croisée entre la fouille de données et la programmation par contraintes [14, 17].

Dans cet article, nous nous intéressons à la mise en œuvre en SAT de notre langage de contraintes. Pour cela, nous présentons l'encodage réalisé et nous montrons comment il tire parti des caractéristiques des solveurs SAT : clauses binaires, propagation unitaire, réseaux de tri, ...Les expérimentations effectuées avec `MiniSat` sur différents jeux de données de l'UCI montrent la faisabilité et l'intérêt de notre approche.

La section 2 rappelle très brièvement les caractéristiques de notre langage de contraintes qui sont utilisées dans cet article. Les apports de notre approche par raf-

finements successifs sont illustrés au travers de deux types de clustering : le clustering conceptuel (cf section 3.2) et le clustering exploitant les connaissances a priori du domaine (cf section 3.3). La section 4 présente l'encodage réalisé. La section 5 montre la faisabilité et l'intérêt de notre approche aux travers de différentes expérimentations menées avec `MiniSat`. La section 6 dresse un bref état de l'art des rares travaux menés dans ce domaine.

2 Aperçu succinct de notre langage de contraintes

Soit $\mathcal{I}$ un ensemble de n littéraux distincts appelés *items*. Un motif ensembliste d'items (ou *itemset*) est un sous-ensemble non vide de $\mathcal{I}$. Ces motifs forment le langage $\mathcal{L}_\mathcal{I} = 2^\mathcal{I} \backslash \emptyset$. Un *contexte transactionnel* $\mathcal{T}$ est défini comme un multi-ensemble de m motifs de $\mathcal{L}_\mathcal{I}$. Chacun de ces motifs, appelé transaction, constitue une entrée de la base de données.

Les contraintes sont des relations portant sur des termes, ces termes étant construits en utilisant des constantes, des variables, des opérateurs et des symboles de fonction. L'utilisateur peut définir de nouveaux symboles de fonctions aussi bien que de nouvelles contraintes.

Les termes sont construits à partir de :
- de constantes qui peuvent être des valeurs numériques, des motifs ou des transactions,
- de variables, notées X_j, pour $1 \leq j \leq k$, représentant les motifs inconnus,
- d'opérateurs ensemblistes (e.g., $\cap, \cup, \setminus$) ou numériques (e.g., $+, -, \times, /$),
- de symboles de fonction portant sur un ou plusieurs motifs tels que :
  - $\texttt{cover}(X_j) = \{t \mid t \in \mathcal{T}, X_j \subseteq t\}$ est le support de X_j, i.e. l'ensemble des transactions couvertes par X_j.
  - $\texttt{freq}(X_j) = \mid \{t \mid t \in \mathcal{T}, X_j \subseteq t\} \mid$ est la fréquence du motif X_j.
  - $\texttt{size}(X_j) = \mid \{i \mid i \in \mathcal{I}, i \in X_j\} \mid$ est la cardinalité du motif X_j.
  - $\texttt{overlapItems}(X_i, X_j) = \mid X_i \cap X_j \mid$ est le nombre d'items communs à X_i et X_j.
  - $\texttt{overlapTransactions}(X_i, X_j) = \mid \texttt{cover}(X_i) \cap \texttt{cover}(X_j) \mid$ est le nombre de transactions couvertes à la fois par X_i et X_j.

Les contraintes sont des relations portant sur les termes. On distingue trois types de contraintes prédéfinies : les contraintes numériques (e.g., $<, \leq, =, \neq, \geq, >$), les contraintes ensemblistes (e.g., $=, \neq, \in, \notin, \subset, \subseteq$), et les contraintes spécifiques telles que :
- $\texttt{isEmpty}(X_j)$ est satisfaite ssi $X_j \neq \emptyset$
- $\texttt{closed}(X_j)$ est satisfaite ssi X_j est un motif fermé [1].
- $\texttt{coverTransactions}([X_1, ..., X_k])$ est satisfaite ssi chaque transaction est couverte par au moins un motif ($\bigcup_{1 \leq i \leq k} \texttt{cover}(X_i) = \mathcal{T}$)
- $\texttt{noOverlapTransactions}([X_1, ..., X_k])$ est satisfaite ssi $\forall i, j$ t.q. $1 \leq i < j \leq k$, $\texttt{cover}(X_i) \cap \texttt{cover}(X_j) = \emptyset$
- $\texttt{coverItems}([X_1, ..., X_k])$ est satisfaite ssi chaque item appartient à au moins un motif ($\cup_{1 \leq i \leq k} X_i = \mathcal{I}$)
- $\texttt{noOverlapItems}([X_1, ..., X_k])$ est satisfaite ssi $\forall i, j$ t.q. $1 \leq i < j \leq k$, $X_i \cap X_j = \emptyset$
- $\texttt{canonical}([X_1, ..., X_k])$ est satisfaite ssi $\forall i$ t.q. $1 \leq i < k$, le motif X_i est inférieur au motif X_{i+1} par rapport à l'ordre lexicographique.

Enfin, une requête est une conjonction de contraintes. Pour un aperçu complet du langage, se reporter à [16].

3 Clustering par raffinements successifs

Un atout majeur de notre approche est de fournir à l'utilisateur un cadre flexible et efficace pour modéliser et raffiner ses requêtes. En pratique, l'utilisateur commence par écrire une requête initiale Q_1 qu'il peut raffiner par des opérations successives d'ajout/retrait de contraintes (Q_{i+1} est dérivée à partir de Q_i) jusqu'à ce l'information extraite soit considérée comme suffisamemt relevante. Nous illustrons cette approche par raffinements successifs grâce à deux exemples de clustering à base d'associations : le clustering conceptuel (cf section 3.2) et le clustering exploitant les connaissances du domaine (cf section 3.3).

3.1 Modélisation d'une requête de clustering

Un problème de clustering peut être décrit par un scénario dans lequel l'utilisateur cherche à obtenir une partition $\pi_T = (X_1, ..., X_k)$ d'une base de données $\mathcal{T}$, contenant m points, en k clusters (non vides). Notre langage de contraintes permet de définir $\texttt{isPartition}([X_1,...,X_k])$ vérifiant que les clusters $(X_1,...,X_k)$ constituent une partition de $\mathcal{T}$:

$$\texttt{isPartition}([X_1, ..., X_k]) \equiv \left\{ \begin{array}{l} \wedge_{1 \leq i \leq k} \texttt{notEmpty}(X_i) \wedge \\ \texttt{coverInstances}([X_1, ..., X_k]) \wedge \\ \texttt{noOverlapInstances}([X_1, ..., X_k]) \end{array} \right.$$

Un problème de clustering contient naturellement des solutions symétriques. Soit $s = (\pi_1, ..., \pi_k)$ une solution contenant k clusters π_i. Toute permutation de

1. soit $Tr_j = \texttt{cover}(X_j)$. X_j est fermé ssi X_j est le plus grand ($\subset$) motif couvrant Tr_j.

ces k motifs est aussi une solution. La contrainte `canonical`$([X_1, ..., X_k])$ est alors utilisée pour éviter les solutions symétriques. Elle impose que pour tout i t.q. $1 \leq i < k$, le cluster X_i est inférieur au cluster X_{i+1} par rapport à l'ordre lexicographique. Nous pouvons ainsi définir `isClustering`$([X_1, ..., X_k])$ qui évite les solutions symétriques :

$$\texttt{isClustering}([X_1, ..., X_k]) \equiv \begin{cases} \texttt{isPartition}([X_1, ..., X_k]) \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{cases}$$

3.2 Clustering conceptuel

Soit $\mathcal{I}$ un ensemble de n items et $\mathcal{T}$ un ensemble de m transactions. Les motifs fermés sont de bons candidats pour la recherche de clusterings à base d'associations. En effet, ils rassemblent un maximum de similitude entre un ensemble de transactions. Un problème de clustering standard peut être formulé comme suit : trouver un ensemble de k motifs fermés $X_1, X_2, ..., X_k$ (i.e., clusters) couvrant toutes les transactions sans chevauchement des supports respectifs de ces motifs (requête Q_1).

$$\texttt{isConceptualCl}([X_1, ..., X_k]) \equiv \begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{closed}(X_i) \end{cases}$$

Eviter les clusters peu fréquents. Quand un cluster extrait est de faible fréquence, il est considéré comme peu représentatif et donc peu fiable. Une contrainte de fréquence minimale (par rapport à un seuil δ_1) peut alors être ajoutée.

$$\begin{cases} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{freq}(X_i) \geq \delta_1 \end{cases}$$

Eviter les clusters de petite taille. Un clustering contenant au moins un motif X_i de petite taille [2] n'est pas considéré comme pertinent parce que X_i ne garantit pas suffisamment de similarité entre les transactions associées. Une contrainte de cardinalité minimale (par rapport à un seuil δ_2) peut alors être ajoutée (requête Q_2).

$$\begin{cases} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{freq}(X_i) \geq \delta_1 \wedge \\ \wedge_{1 \leq i \leq k}\ \texttt{size}(X_i) \geq \delta_2 \end{cases}$$

Equilibrage des clusters. Dans les problèmes de clustering conceptuel, on a généralement tendance à préférer les solutions dans lesquelles les fréquences des différents clusters extraits sont proches les unes des autres. Pour chaque couple de clusters (X_i, X_j), leur différence de fréquence doit être inférieure à un seuil $\Delta \times m$ où Δ est un pourcentage (requête Q_3).

$$\begin{cases} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \mid \texttt{freq}(X_i) - \texttt{freq}(X_j) \mid \leq \Delta \times m \end{cases}$$

D'une manière analogue, il est possible de modéliser d'autres problèmes de clustering [5] tels que le soft clustering, le co-clustering, ou encore le soft co-clustering.

Soft clustering est une version relaxée du problème de clustering dans lequel on autorise un certain chevauchement entre les supports des motifs extraits (inférieur à un seuil δ_T).

$$\begin{cases} \wedge_{1 \leq i \leq k}\ \texttt{closed}(X_i) \wedge \\ \texttt{coverTransactions}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \texttt{overlapTransactions}(X_i, X_j) \leq \delta_T \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{cases}$$

Co-clustering vise à trouver k clusters couvrant à la fois l'ensemble des transaction et l'ensemble des items, sans chevauchement ni entre les motifs ni entre leurs supports.

$$\begin{cases} \texttt{isConceptualCl}([X_1, ..., X_k]) \wedge \\ \texttt{coverItems}([X_1, ..., X_k]) \wedge \\ \texttt{noOverlapItems}([X_1, ..., X_k]) \end{cases}$$

Soft co-clustering est une version relaxée du co-clustering dans lequel on autorise un certain chevauchement entre les motifs extraits (inférieur à un seuil δ_I) et un certain chevauchement entre leurs supports (inférieur à un seuil δ_T).

$$\begin{cases} \wedge_{1 \leq i \leq k}\ \texttt{closed}(X_i) \wedge \\ \texttt{coverTransactions}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \texttt{overlapTransactions}(X_i, X_j) \leq \delta_T \wedge \\ \texttt{coverItems}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k}\ \texttt{overlapItems}(X_i, X_j) \leq \delta_I \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{cases}$$

3.3 Clustering exploitant les connaissances a priori

Lors d'applications réelles, l'utilisateur possède souvent des connaissances a priori (portant sur le domaine ou la base de données) qui peuvent être utiles pour faire émerger des groupes de données. Certaines méthodes de clustering ne savent pas tirer parti de ces connaissances [3, 5]. De telles connaissances sont souvent exprimées par des contraintes portant sur les transactions (e.g., les contraintes `mustLink` et `cannotLink` [19, 20]), et des contraintes portant sur les clusters (e.g., les contraintes de *Diamètre Maximal* et de *Séparation Minimale* [7, 8]).

Dans cette section, nous décrivons comment ces contraintes peuvent être modélisées en utilisant notre langage de contraintes. Puis, nous montrons comment

2. Les motifs de taille 1 par exemple, reflètent souvent des valeurs qui ne sont apparues que lors de la binarisation d'un attribut.

elles peuvent être combinées pour obtenir des requêtes plus pertinentes imposant des tailles minimales de clusters ou cherchant des clusterings équilibrés. Soit $\mathcal{T}$ une base de données composée de m transactions. Soit $d(t_1, t_2)$ une distance définie sur les transactions.

Contraintes au niveau des transactions :

- $\texttt{mustLink}(t_1, t_2)$ impose que les transactions t_1 et t_2 appartiennent au même cluster.
- $\texttt{cannotLink}(t_1, t_2)$ impose que les transactions t_1 et t_2 n'appartiennent pas au même cluster.

Contrainte de diamètre maximal. Le *diamètre d'un cluster* X_j est la distance maximale entre toute paire de transactions appartenant au support de X_j. La contrainte de diamètre maximal impose que le diamètre de chaque cluster soit au plus égal à une valeur donnée α. Pour toute paire de transactions (t_i, t_j) t.q. $d(t_i, t_j) > \alpha$, t_i et t_j ne pourront appartenir à un même cluster et devront satisfaire la contrainte $\texttt{cannotLink}(t_i, t_j)$.

Contrainte de séparation minimale. La *séparation entre deux clusters* X_i *et* X_j est la distance minimale entre toute paire de transactions, l'une appartenant au support de X_i et l'autre au support de X_j. La contrainte de séparation minimale impose que la séparation entre deux clusters soit au moins égale à une valeur donnée β. Pour toute paire de transactions (t_i, t_j) t.q. $d(t_i, t_j) < \beta$, t_i et t_j devront appartenir au même cluster et satisfaire la contrainte $\texttt{mustLink}(t_i, t_j)$.
Ces différents types de contraintes peuvent être combinés (requête Q'_1).

$$\begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \end{cases}$$

Des connaissance a priori sur des transactions et/ou clusters donnés peuvent aisément être modélisées. L'appartenance (resp. la non appartenance) de deux transactions t_{i_0} et t_{j_0} données au même cluster est modélisée par l'ajout de la contrainte : $\texttt{mustLink}(t_{i_0}, t_{j_0})$ (resp. $\texttt{cannotLink}(t_{i_0}, t_{j_0})$). L'appartenance (resp. la non appartenance) d'une transaction t_{i_0} donnée au cluster X_{j_0} est modélisée par l'ajout de la contrainte : $t_{i_0} \in X_{j_0}$ (resp. $t_{i_0} \notin X_{j_0}$).

Affiner les requêtes. D'une manière analogue au clustering conceptuel, les clusters de petite taille sont considérés comme non-pertinents et peuvent être ignorés (requête Q'_2).

$$\begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i \leq k} \texttt{size}(X_i) \geq \delta \wedge \end{cases}$$

Et rechercher des clusterings équilibrés (requête Q'_3).

$$\begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq k} \mid \texttt{size}(X_i) - \texttt{size}(X_j) \mid \leq \Delta \times m \end{cases}$$

4 Encodage sous la forme d'une CNF

Après avoir sélectionné le jeu de données et avoir modélisé la requête, il est nécessaire de traduire l'ensemble sous forme d'une CNF qui sera composée :

- de la CNF associée à la recherche des différents motifs inconnus (cf sections 4.1 et 4.2) ;
- des CNFs associées à l'encodage de chaque contrainte figurant dans la requête (cf sections 4.3 à 4.5).

Le clustering conceptuel, où chaque cluster est représenté par un motif fermé, requiert un encodage plus volumineux que le clustering avec connaissances a priori (cf sections 4.1 et 4.2). L'encodage des contraintes de partitionnement est le même dans les deux cas (cf section 4.3). Les contraintes de seuil sont encodées grâce à des réseaux de tri, qui permettent un encodage indépendant de la valeur du seuil (cf section 4.5). Finalement, nous montrons comment les propriétés d'inférence des contraintes `mustLink` et `cannotLink` sont automatiquement déduites par le solveur SAT (cf section 4.6) et comment ce dernier peut être est rendu complet grâce à l'utilisation de *restarts* (cf section 4.7).

4.1 Encodage du clustering conceptuel

Soit $\mathcal{I}$ un ensemble de n items et $\mathcal{T}$ un ensemble de m transactions. Notons $(d_{t,i})$ la matrice booléenne de taille (m, n) telle que $(d_{t,i}$=`True`$)$ ssi $(i \in t)$. Les motifs inconnus sont modélisés grâce à des variables booléennes et à cette matrice $(d_{t,i})$.

Soit $X_1, X_2, ..., X_k$ un ensemble de k motifs inconnus. De la même façon que [14, 17], le lien entre le jeu de données et les motifs inconnus X_j est établi à l'aide de deux ensembles de variables :

- $X_{1,j}, X_{2,j}, ..., X_{n,j}$ t.q. $(X_{i,j}$=`True`$)$ ssi $(i \in X_j)$
- $T_{1,j}, T_{2,j}, ..., T_{m,j}$ t.q. $(T_{t,j}$=`True`$)$ ssi $(X_j \subset t)$

Considérons le motif inconnu X_j, la relation de couverture $(X_j \subset t)$ peut alors être modélisée par la CNF suivante :

$$\bigwedge_{\{i \in \mathcal{I} \mid \neg d_{t,i}\}} \neg X_{i,j} \tag{1}$$

La relation entre X_j et $\mathcal{T}$ est modélisée en établissant que, pour chaque transaction t, $(T_{t,j}$=`True`$)$ ssi X_j couvre t :

$$\forall t \in \mathcal{T}, T_{t,j} \Leftrightarrow (X_j \subset t) \tag{2}$$

En utilisant Eq. 1, l'implication (gauche-droite) de Eq. 2 peut être modélisée par la CNF suivante :

$$\bigwedge_{t\in\mathcal{T}}(\bigwedge_{\{i\in\mathcal{I}|\neg d_{t,i}\}}(\neg T_{t,j}\vee\neg X_{i,j})) \quad (3)$$

En utilisant Eq. 1, l'implication (droite-gauche) de Eq. 2 peut être modélisée par la CNF suivante :

$$\bigwedge_{t\in\mathcal{T}}((\bigvee_{\{i\in\mathcal{I}|d_{t,i}\}}X_{i,j})\vee T_{t,j}) \quad (4)$$

Enfin, Eq. 3 et Eq. 4 doivent être vérifiées pour tout motif X_j. L'encodage nécessite donc $(k\times m\times n)$ clauses binaires.
La contrainte $\texttt{closed}(X_j)$ est modélisée [3] par la formule suivante :

$$\bigwedge_{i\in\mathcal{I}}(X_{i,j}\Leftrightarrow\bigwedge_{\{t\in\mathcal{T}|\neg d_{t,i}\}}\neg T_{t,j})$$

4.2 Encodage du clustering avec connaissances a priori

Soit $\mathcal{T}$ un ensemble de m transactions. Chaque cluster inconnu est modélisé à l'aide de m variables booléennes $T_{t,j}$ telles que $(T_{t,j}=True)$ ssi la transaction t appartient au cluster j. Un cluster est désigné ici par son index (compris entre 1 et k) et non plus par un motif fermé. C'est pourquoi, l'encodage de ce type de clustering est de plus petite taille que celui du clustering conceptuel.

4.3 Encodage des contraintes de partitionnement

L'encodage de ces contraintes est le même pour les deux types de clustering car il n'utilise que les variables $T_{t,j}$.

- $\texttt{coverTransactions}([X_1,...,X_k])$ impose que chaque transaction $t\in\mathcal{T}$ appartienne à au moins un cluster. Ainsi, pour chaque $t\in\mathcal{T}$, il existe au moins un cluster X_j t.q. t figure dans X_j.

$$\bigwedge_{t\in\mathcal{T}}(\bigvee_{j\in[1..k]}T_{t,j})$$

- $\texttt{noOverlapTransactions}([X_1,...,X_k])$ s'assure que chaque transaction $t\in\mathcal{T}$ appartient à au plus un cluster. Une transaction t appartient à au moins deux clusters ssi il existe X_i et X_j t.q. $(t\in X_i)\wedge(t\in X_j)$, c'est-à-dire $\bigvee_{1\leq i<j\leq k}(T_{t,i}\wedge T_{t,j})$. Il ne reste plus qu'à prendre la négation de cette formule pour chaque $t\in\mathcal{T}$.

$$\bigwedge_{t\in\mathcal{T}}(\bigwedge_{1\leq i<j\leq k}(\neg T_{t,i}\vee\neg T_{t,j}))$$

- $\texttt{notEmpty}(X_j)$ impose qu'il existe au moins une transaction $t\in\mathcal{T}$ appartenant au cluster X_j et est modélisée par la clause :

$$\bigvee_{t\in\mathcal{T}}T_{t,j}$$

- $\texttt{canonical}([X_1,...,X_k])$ impose que pour tout i t.q. $1\leq i<k$, X_i soit inférieur à X_{i+1} et est encodée grâce à un comparateur binaire.

4.4 Encodage des contraintes mustLink et cannotLink

$\texttt{mustLink}(t_1,t_2)$ contraint deux transactions t_1 et t_2 à appartenir à un même cluster. Ainsi, pour chaque $j\in[1..k]$, $T_{t_1,j}\Leftrightarrow T_{t_2,j}$.

$$\bigwedge_{1\leq j\leq k}(\neg T_{t_1,j}\vee T_{t_2,j})\wedge(T_{t_1,j}\vee\neg T_{t_2,j})$$

$\texttt{cannotLink}(t_1,t_2)$ contraint deux transactions t_1 et t_2 à ne pas appartenir au même cluster. Ainsi, pour chaque $j\in[1..k]$, $T_{t_1,j}\Leftrightarrow\neg T_{t_2,j}$.

$$\bigwedge_{1\leq j\leq k}(\neg T_{t_1,j}\vee\neg T_{t_2,j})\wedge(T_{t_1,j}\vee T_{t_2,j})$$

Notons que ces contraintes sont encodées en utilisant $(2\times k)$ clauses binaires.

4.5 Encodage des contraintes de seuil à l'aide de réseaux de tri

Les contraintes de seuil sont directement modélisées à l'aide de contraintes de cardinalité portant sur les variables booléennes définies aux sections 4.1 et 4.2. Par exemple, $\texttt{freq}(X_j)\geq\ \delta_1$ se modélise $\#(T_{1,j},T_{2,j},...,T_{m,j})\geq\delta_1$, et $\texttt{size}(X_j)\geq\delta_2$ se modélise $\#(X_{1,j},X_{2,j},...,X_{n,j})\geq\delta_2$. Il en est de même pour les autres contraintes de seuil.

Plusieurs encodages performants des contraintes de cardinalité ont été proposés [1, 18]. Ces travaux utilisent des additionneurs unaires permettant d'établir la cohérence d'arc via la propagation unitaire. Mais de tels encodages requièrent une nombre quadratique de clauses [1] ou dépendent de la valeur du seuil [18] (plus grand est le seuil, plus grande est la CNF résultante). Dans le cas du clustering, ces seuils sont souvent très grands, et la taille de l'encodage devient rapidement prohibitive. C'est pourquoi nous avons choisi d'utiliser les réseaux de tri pour encoder les contraintes de cardinalité. Tout d'abord, la taille de cet encodage ne dépend plus de la valeur de seuil. De plus, l'utilisation de réseaux de tri permet aussi d'établir la cohérence d'arc via la propagation unitaire [11].

3. Soit Tr_j=$\texttt{cover}(X_j)$. X_j est un motif fermé (i.e. le plus grand motif couvrant Tr_j) ssi $\forall i\in\mathcal{I}, i\in X_j\Leftrightarrow\forall t\in\mathcal{T}, i\notin t\Rightarrow X_j\not\subset t$. Cette modélisation repose sur les propriétés de la connexion de Galois (voir [17] pour plus de détails).

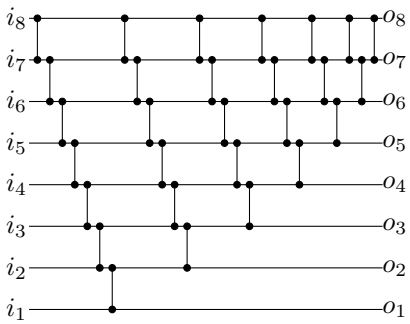


FIGURE 1 – Réseau de tri bulles (n=8).

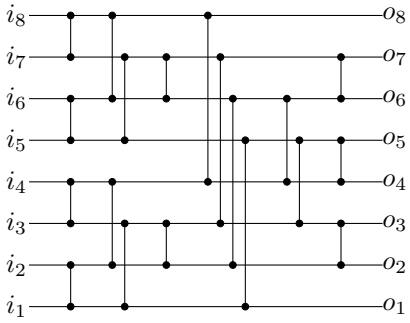


FIGURE 2 – Réseau de tri *Batcher* (n=8).

Les réseaux de tri peuvent être utilisés pour compter le nombre de littéraux positifs. Pour n entrées, le réseau associe n sorties qui correspondent à une version triée des entrées. Cette sortie triée constitue une représentation unaire du nombre de littéraux positifs. Les réseaux de tri utilisent une unique opération d'échange binaire et l'appliquent plusieurs fois afin de trier complètement les n littéraux en entrée. Un échange binaire prend en entrée deux littéraux i_1 et i_2, et retourne deux littéraux o_1 et o_2 tel que o_1 est le minimum entre i_1 i_2 ($o_1 \Leftrightarrow i_1 \wedge i_2$) et o_2 leur maximum ($o_2 \Leftrightarrow i_1 \vee i_2$).

Plusieurs algorithmes de tri ont été implémentés à l'aide des réseaux de tri. La figure 1 représente le réseau associé au tri bulles pour n=8 entrées. Chaque ligne horizontale représente un littéral tout au long du processus de tri, et chaque segment vertical représente un échange binaire entre les deux littéraux à ses extrémités.

Les réseaux de tri fournissent seulement une représentation unaire du nombre de littéraux positifs. Pour modéliser une contrainte de seuil, il est nécessaire de contraindre les littéraux en sortie du réseau. Ainsi, pour contraindre au moins δ littéraux à être positifs, le littéral o_δ doit être instancié à `True`. De la même façon, pour contraindre au plus δ littéraux à être positifs, le littéral $o_{\delta+1}$ doit être instancié à `False`.

L'utilisation d'un tri naïf, comme le tri bulles, peut conduire à un grand nombre d'échanges et donc à un encodage trop volumineux ($O(n^2)$). Nous avons choisi d'utiliser le tri *odd-even Batcher sort* [4] qui requiert moins d'échanges ($O(n \times (log\ n)^2)$). La figure 2 représente le réseau de tri associé à cet algorithme de tri pour n=8. Ce réseau de tri a déjà été utilisé dans `MiniSat+` et s'est avéré très efficace par rapport aux autres encodages des contraintes de cardinalité [11].

4.6 Transitivité des `mustLink` et `cannotLink`

Soit G=(V, E) un graphe constitué de contraintes `mustLink` où V=$\mathcal{T}$ et il existe une arête entre t_i et t_j ssi il existe une contrainte `mustLink`(t_i, t_j) [3, 20]. Soit CC_1 et CC_2 deux composantes connexes de G. *(i) s'il existe une contrainte* `mustLink`(t_1, t_2) *t.q.* $t_1 \in CC_1$ *et* $t_2 \in CC_2$*, alors il est possible d'inférer les contraintes* `mustLink`(x, y) *pour tout* $x \in CC_1$ *et tout* $y \in CC_2$*.* À l'inverse de `mustLink`, `cannotLink` n'est pas une relation d'équivalence, mais *(ii) s'il existe une contrainte* `cannotLink`(t_1, t_2) *t.q.* $t_1 \in CC_1$ *et* $t_2 \in CC_2$*, alors il est possible d'inférer les contraintes* `cannotLink`(x, y) *pour tout* $x \in CC_1$ *et tout* $y \in CC_2$*.*

De telles déductions sont habituellement faites en ajoutant toutes les contraintes `mustLink` et `cannotLink` ainsi inférées [3, 20]. Mais, grâce à l'utilisation de notre encodage (cf. section 4.4), il n'est plus nécessaire d'effectuer ces ajouts. En effet, toutes les contraintes déduites seront implicitement prises en compte par le solveur SAT.

4.7 Établir la complétude

Pour une CNF donnée, un solveur SAT, soit retourne une instanciation des variables (et seulement une) telle que la CNF est évaluée à `True`, soit prouve qu'il n'existe aucune instanciation la satisfaisant. Afin d'assurer la complétude de notre approche, plusieurs *restarts* sont réalisés. La CNF $\mathcal{F}$ modélisant la requête soumise est transmise au solveur. Après avoir obtenue une solution s_i, la négation de cette solution $\neg s_i$ est ajoutée à $\mathcal{F}$. Ce processus est itéré tant que le solveur SAT trouve de nouvelles solutions. L'utilisation de *restarts* peut sembler naïve, mais est en pratique très efficace (cf section 5). Cela est notamment dû au fait que $\mathcal{F}$ contienne un très grand nombre de clauses binaires qui permettent un filtrage par propagation unitaire très efficace.

5 Expérimentations

Nous avons utilisé le solveur `MiniSat` [4] [10] pour effectuer différentes expérimentations sur plusieurs jeux de données de l'UCI [5] ainsi qu'un jeu de données réelles

4. `http://minisat.se/`
5. `http://www.ics.uci.edu/~mlearn/MLRepository.html`

Jeu de données	m	n	Densité
`Australian`	653	125	0.40
`Mushroom`	8124	119	0.19
`P.-Tumor`	336	36	0.48
`Soybean`	630	50	0.32
`Zoo`	101	36	0.44
`Meningitis`	329	82	0.26

TABLE 1 – Caractéristiques des jeux de données.

nommé `Meningitis` et provenant de l'Hôpital Central de Grenoble. `Meningitis` recense les pathologies des enfants atteints d'une méningite virale ou bactérienne. La table 1 résume les différentes caractéristiques de ces jeux de données. La densité d'un jeu de données est $d{=}\sum d_{t,i}/(n \times m)$ (cf section 4.1).

Le processeur utilisé est un Core2Duo E8400 (2.83GHz) avec 4Go de RAM. Pour chaque expérimentation, nous avons reporté les temps CPU nécessaires au calcul de la première et des dix premières solutions en fonction du nombre k de clusters souhaités dans la requête. Le symbole `'-'` indique l'absence de solutions pour une requête.

5.1 Clustering conceptuel

La figure 3 indique les temps CPU nécessaires au calcul de la première et des dix premières solutions pour la requête Q_1 (cf section 3.2), ceci pour différentes valeurs de k. Notre approche est efficace, même pour des valeurs de k relativement grandes, et des jeux de données possédant un très grand nombre de motifs fermés (`Australian` possède plus de 20 millions de motifs fermés).

La figure 4 indique les temps CPU pour la requête Q_2 avec $\delta_1{=}m/10$ et $\delta_2{=}2$ (cf section 3.2). Les temps CPU augmentent car Q_2 est plus contrainte que Q_1 et comporte des contraintes de seuil. `Mushroom` est le jeu de données de plus grande taille. Il requiert une très grande quantité de clauses pour encoder les contraintes de seuil (1.4 million par motif inconnu [6]). Pour $k{=}10$, il n'y a de solution pour aucun jeu de données. En effet, il est impossible de trouver un clustering constitué de 10 motifs fermés différents ayant une fréquence supérieure à $m/10$ puisqu'aucun recouvrement n'est autorisé entre les m transactions.

La figure 5 indique les temps CPU nécessaires au calcul de la première solution pour la recherche d'un clustering équilibré (requête Q_3). Le ratio Δ est fixé à 20% (cf partie gauche de la figure 5) et à 40% (cf partie droite de la figure 5). Les temps CPU augmentent mais demeurent raisonnables.

La figure 6 décrit l'évolution du temps CPU en fonction du nombre de solutions pour le jeu de données `P.-Tumor` avec $k{=}6$. Une courbe est associée à chacune des trois requêtes Q_1 (rouge), Q_2 (bleu) et Q_3 avec $\Delta{=}40\%$ (noir). Il est à noter que la requête Q_3 possède seulement 18 solutions. Les temps CPU pour Q_2 et Q_3 sont plus élevés que pour Q_1 mais demeurent abordables pour ce jeu de données. La figure 7 décrit la même évolution pour `Australian` avec $k{=}6$. Ce jeu de données est le pire compte-tenu du très grand nombre de motifs fermés qu'il contient.

La figure 8 indique la taille des encodages [7] des requêtes Q_1, Q_2 et Q_3 pour les jeux de données `Australian` et `Mushroom`. Même si l'encodage peut être de grande taille (plusieurs millions de clauses), le ratio r de clauses binaires demeure toujours très élevé.

5.2 Clustering exploitant les connaissances a priori

Ces expérimentations ont été menées sur les mêmes jeux de données que pour le clustering conceptuel (cf table 1) en utilisant la distance de Hamming entre deux transactions [8]. Le diamètre maximal α a été fixé à $n/20$ et la séparation minimale β à $n/2$.

La figure 9 indique les temps CPU nécessaires au calcul de la première et des dix premières solutions pour la requête Q'_1 (cf section 3.3), ceci pour différentes valeurs de k. Pour chaque jeu de données, les dix premières solutions (si elles existent) sont obtenues très rapidement, y compris pour `Mushroom` qui est le jeu de données de plus grande taille.

La figure 10 indique les temps CPU nécessaires au calcul de la première solution pour la recherche d'un clustering équilibré (requête Q'_3 de la section 3.3). Le ratio Δ a été fixé à 10% (cf partie gauche de la figure 10) et à 20% (cf partie droite de la figure 10). Même avec des contraintes additionnelles de seuil, les temps CPU demeurent raisonnables. Enfin, ces seuils étant élevés, peu de requêtes possèdent (au moins) une solution.

La figure 11 décrit l'évolution du temps CPU en fonction du nombre de solutions pour le jeu de données `Zoo` avec $k{=}6$. Une courbe est associée à chacune des trois requêtes Q'_1 (rouge), Q'_2 avec $\delta{=}m/10$ (bleu) et Q'_3 avec $\Delta{=}20\%$ (noir). Ces courbes semblent être quasi-linéaires. Chacune des trois requêtes comporte de nombreuses contraintes `mustLink` et `cannotLink` qui sont encodées à l'aide de clauses binaires. Dès qu'une transaction est affectée à un cluster, de nombreuses déductions vont être effectuées grâce à la propagation unitaire et à la transitivité de ces contraintes (cf section 4.6). Mais, cette justification se doit d'être vérifiée de manière plus précise par de nouvelles expérimentations. La figure 12 donne des résultats similaires pour le jeu de données `Australian` ($k{=}6$).

6. C'est 50 fois moins que l'encodage proposé par Bailleux et 10 fois moins que celui proposé par Sinz (cf section 4.5).

7. Une unité représente 1 millier de litéraux ou de clauses.

8. Toute autre distance aurait pu être utilisée car celle-ci ne sert qu'à imposer les contraintes `mustLink` et `cannotLink`.

jeu de données	k=2	k=4	k=6	k=8	k=10
`Australian`	0.03	0.06	0.41	12.95	132.3
`Mushroom`	0.47	3.86	10.95	58.32	25.58
`P.-Tumor`	0.01	0.02	0.05	0.05	0.08
`Soybean`	0.01	0.07	1.05	1.04	1.09
`Zoo`	0.01	0.01	0.02	0.02	0.06
`Meningitis`	0.02	0.04	0.28	3.30	9.76

k=2	k=4	k=6	k=8	k=10
0.04	0.69	1.67	27.21	190.2
-	14.17	31.65	123.9	134.7
0.01	0.03	0.06	0.22	0.79
-	0.19	2.36	7.04	4.36
0.01	0.01	0.02	0.03	0.08
0.03	0.22	5.28	7.26	20.19

FIGURE 3 – (Q_1) Temps en s. pour la 1ère sol. (gauche) et les 10 premières sol. (droite).

jeu de données	k=2	k=4	k=6	k=8	k=10
`Australian`	0.18	1.35	305.8	2233	-
`Mushroom`	3.39	58.73	2715	-	-
`P.-Tumor`	-	0.09	4.55	104.3	-
`Soybean`	-	1.27	72.27	395.1	-
`Zoo`	0.02	0.02	0.10	0.51	-
`Meningitis`	-	4.14	186.5	1523	-

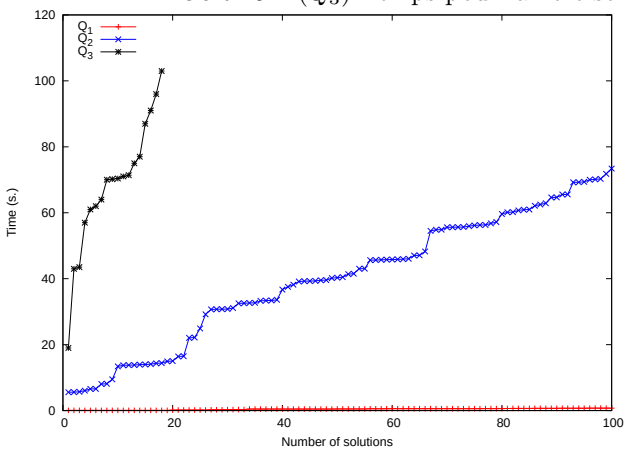
k=2	k=4	k=6	k=8	k=10
0.40	10.38	464.1	21856	-
-	114.9	10125	-	-
-	0.64	11.66	234.6	-
-	7.76	120.5	824.0	-
-	0.06	0.28	2.61	-
-	29.44	664.8	5748	-

FIGURE 4 – (Q_2) Temps en s. pour la 1ère sol. (gauche) et les 10 premières sol. (droite).

jeu de données	k=2	k=4	k=6	k=8	k=10
`Australian`	0.14	31.1	909.5	13274	-
`Mushroom`	-	-	-	-	-
`P.-Tumor`	0.04	0.94	-	66.45	900.7
`Soybean`	-	-	-	-	-
`Zoo`	-	-	-	3.59	1.31
`Meningitis`	-	-	-	-	-

k=2	k=4	k=6	k=8	k=10
0.12	4.78	407.0	5285	-
4.11	-	-	-	35645
0.04	0.44	18.95	108.5	91.9
0.11	2.37	-	-	1043
0.02	0.11	0.48	0.60	0.52
-	-	-	3730	-

FIGURE 5 – (Q_3) Temps pour la 1ère sol. avec Δ=20% (gauche) et Δ=40% (droite).



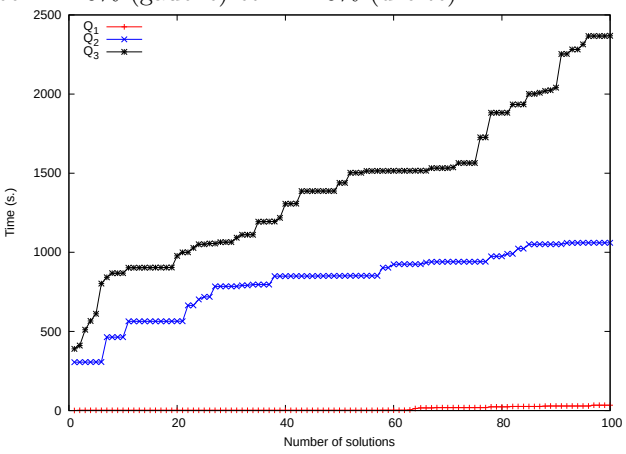


FIGURE 6 – Temps pour `P.-Tumor` (k=6).

`Australian`		$k(=2)$	$(k=6)$	$(k=10)$
Q_1	#littéraux	2.4	6.5	10.6
	#clauses	102.6	315.2	538.3
	ratio %	97.1	97.1	97.2
Q_2	#littéraux	104.6	312.9	521.3
	#clauses	409.1	1235	2071
	ratio %	74.3	74.5	74.6
Q_3	#littéraux	98.7	295.3	491.9
	#clauses	392.6	1185	1988
	ratio %	74.7	74.9	75.0

FIGURE 7 – Temps pour `Australian` (k=6).

`Mushroom`		$(k=2)$	$(k=6)$	$(k=10)$
Q_1	#littéraux	24.8	58.7	92.6
	#clauses	1650	5018	8516
	ratio %	98.5	98.8	98.9
Q_2	#littéraux	1341	4008	6675
	#clauses	5600	16868	28298
	ratio %	76.1	76.2	76.3
Q_3	#littéraux	1335	3990	6646
	#clauses	5598	16863	28291
	ratio %	76.1	76.3	76.5

FIGURE 8 – Nombre de littéraux et de clauses (une unité = 1000).

jeu de données	k=4	k=8	k=12	k=16	k=20
`Australian`	-	0.02	0.25	0.89	1.59
`Mushroom`	0.96	1.38	3.94	6.64	-
`P.-Tumor`	-	0.03	0.08	0.12	0.13
`Soybean`	0.01	0.03	0.12	0.16	-
`Zoo`	0.01	0.01	0.02	0.03	0.03

k=4	k=8	k=12	k=16	k=20
-	0.05	0.31	0.96	1.70
1.28	1.53	5.38	8.24	-
-	0.03	0.11	0.15	0.17
0.02	0.05	0.14	0.18	-
0.01	0.01	0.03	0.04	0.04

FIGURE 9 – (Q'_1) Temps en s. pour la 1ère sol. (gauche) et les 10 premières sol. (droite).

jeu de données	k=4	k=8	k=12	k=16	k=20
`Australian`	-	19.7	27.46	45.23	180.5
`Mushroom`	16.9	-	-	-	-
`P.-Tumor`	-	1.06	2.22	6.16	6.05
`Soybean`	-	-	-	-	-
`Zoo`	0.01	0.09	-	-	-

k=4	k=8	k=12	k=16	k=20
-	7.20	13.10	46.15	69.76
19.6	-	-	-	-
-	0.91	3.30	3.88	6.08
-	-	-	-	-
0.03	0.10	-	-	-

FIGURE 10 – (Q'_3) Temps pour la 1ère sol. avec Δ=10% (gauche) et Δ=20% (droite).

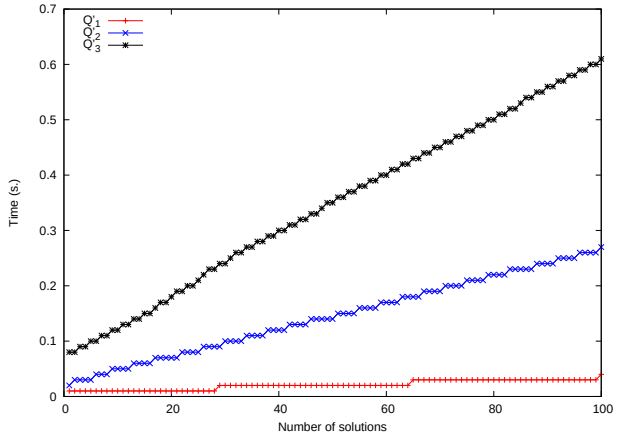


FIGURE 11 – Temps pour `Zoo` (k=6).

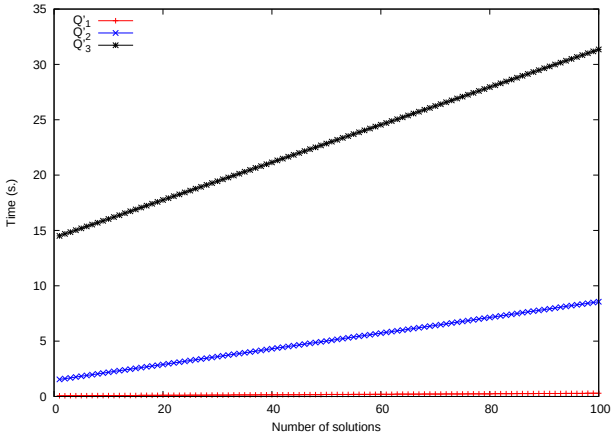


FIGURE 12 – Temps pour `Australian` (k=6).

`Australian`		$(k=2)$	$(k=6)$	$(k=10)$
Q'_1	#littéraux	2.6	10.4	18.2
	#clauses	29.8	107.8	196.3
	ratio %	86.9	81.2	83.93
Q'_2	#littéraux	98.8	299.2	499.5
	#clauses	318.7	974.5	1640
	ratio %	68.6	68.3	68.5
Q'_3	#littéraux	98.8	299.2	499.5
	#clauses	318.7	974.5	1640
	ratio %	68.6	68.3	68.5

`Mushroom`		$(k=2)$	$(k=6)$	$(k=10)$
Q'_1	#littéraux	32.4	129.9	227.4
	#clauses	1141	3651	6291
	ratio %	95.7	93.1	92.9
Q'_2	#littéraux	1343	4062	6781
	#clauses	5075	15452	25960
	ratio %	73.2	72.9	73.1
Q'_3	#littéraux	1343	4062	6781
	#clauses	5075	15452	25960
	ratio %	73.2	72.9	73.1

FIGURE 13 – Nombre de littéraux et de clauses (une unité = 1000).

La figure 13 indique la taille des encodages des requêtes Q'_1, Q'_2 et Q'_3 pour les jeux de données `Australian` et `Mushroom`. La taille de la CNF associée dépend grandement des valeurs α et β. Pour nos expérimentations, le diamètre maximal α a été fixé à $n/20$ et la séparation minimale β à $n/2$. La figure 13 indique aussi le ratio r de clauses binaires dans la CNF. Comme pour le clustering conceptuel (cf figure 8), l'encodage d'une requête peut être volumineux (plusieurs millions de clauses), mais le ratio r de clauses binaires demeure toujours élevé. Enfin, il est à noter que les tailles des CNFs associées à Q'_2 et Q'_3 sont très proches. En effet, seuls les seuils des contraintes de cardinalité sont différents (cf section 4.5).

Pour conclure, ces expérimentations montrent que les solveurs SAT permettent de résoudre de manière efficace ces deux types de clustering. Ils peuvent trouver, dans des temps très compétitifs, les premières solutions (ou prouver qu'il n'en existe pas), même pour des jeux de données difficiles (comme `Australian`) ou de grande taille (comme `Mushroom`).

6 Travaux relatifs

SAT pour le clustering. L'introduction des contraintes sur les transactions (`mustLink` et `cannotLink`) et des contraintes sur les clusters (Diamètre Maximal et Séparation Minimale) dans l'algorithme de clustering `k-means` [15] est décrite dans [7]. Une étude de complexité de ces deux familles de contraintes est présentée dans [8]. Davidson&al ont proposé la première approche utilisant SAT pour le clustering [9], mais elle ne traite que le cas très particulier où k=2, qui conduit

à une modélisation qui est une instance de 2-SAT. Gilpin&Davidson se sont intéressés au clustering hiérarchique [13] et montrent comment des *dendograms* peuvent être modélisés comme des instances du problème Horn-SAT.
SAT pour l'extraction de motifs. Une approche SAT pour résoudre le problème EMS *(Enumerating all Motifs in a Sequence)* a été proposée dans [6] et appliquée à deux problèmes concrets en bio-informatique et en sécurité informatique.

7 Conclusion et perspectives

Nous avons présenté l'encodage SAT de différents problèmes de clustering et nous avons montré comment il tire parti des caractéristiques des solveurs SAT : clauses binaires, propagation unitaire, réseaux de tri, ...Les expérimentations effectuées avec `MiniSat` sur différents jeux de données de l'UCI montrent la faisabilité et l'intérêt de notre approche.

Plusieurs améliorations peuvent être apportées à cet encodage pour en accélérer les performances : heuristiques de sélection de variables, exploitation des *backdoors* et des *nogoods*, ... Le passage à l'échelle, pour des valeurs de k plus grandes, sera aussi traité. Enfin, nous étudierons comment intégrer des critères d'optimisation dans cette approche.

Références

[1] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *CP'03*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.

[2] A. Banerjee and J. Ghosh. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discov.*, 13(3) :365–395, 2006.

[3] S. Basu, I. Davidson, and Kiri L. Wagstaff. *Constrained Clustering : Advances in Algorithms, Theory, and Applications.* Chapman & Hall, 2008.

[4] K. E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.

[5] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA, 2002.

[6] E. Coquery, S. Jabbour, and L. Sais. A constraint programming approach for enumerating motifs in a sequence. In *Workshop on Declarative Pattern Mining, ICDM 2011*, pages 1091–1097, Vancouver, Canada, December 2011.

[7] I. Davidson and S. Ravi. Clustering with constraints : Feasibility issues and the k-means algorithm. In *SDM*, 2005.

[8] I. Davidson and S. Ravi. Using instance-level constraints in agglomerative hierarchical clustering : theoretical and empirical results. *Data Min. Knowl. Discov.*, 18(2) :257–282, 2009.

[9] I. Davidson, S. Ravi, and L. Shamis. A SAT-based framework for efficient constrained clustering. In *SDM*, pages 94–105. SIAM, 2010.

[10] N. Eén and N. Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518, 2003.

[11] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *JSAT*, 2(1-4) :1–26, 2006.

[12] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2) :139–172, 1987.

[13] S Gilpin and I. Davidson. Incorporating SAT solvers into hierarchical clustering algorithms : an efficient and flexible approach. In *KDD'11*, pages 1136–1144, 2011.

[14] M. Khiari, P. Boizumault, and B. Crémilleux. Constraint programming for mining n-ary patterns. In *CP'10*, volume 6308 of *LNCS*, pages 552–567, 2010.

[15] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[16] J.-P. Métivier, P. Boizumault, B. Crémilleux, M. Khiari, and S. Loudni. A constraint-based language for declarative pattern discovery. In *27th annual ACM Symposium on Applied Computing (SAC'12)*, pages 438–444, March 2012.

[17] L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *KDD'08*, pages 204–212, 2008.

[18] C. Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *CP'05*, volume 3709 of *LNCS*, pages 827–831. Springer, 2005.

[19] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In P. Langley, editor, *ICML'00*, pages 1103–1110. M. Kaufmann, 2000.

[20] K. Wagstaff, C. Cardie, S. Rogers, and St. Schrödl. Constrained k-means clustering with background knowledge. In *ICML'01*, pages 577–584, 2001.