BoolLoader : un chargeur efficace dédié aux bases de transactions denses

Zahir Maazouzi[1] Ansaf Salleb Christel Vrain

LIFO Rue Léonard de Vinci, BP 6759, 45067 Orléans Cedex 02
{salleb, cv}@lifo.univ-orleans.fr

Résumé. Nous nous intéressons à la représentation et au chargement de bases de transactions en mémoire. Pour cela, nous proposons d'utiliser un format condensé fondé sur les diagrammes de décision binaires et nous présentons un algorithme que nous avons implanté en un système baptisé *BoolLoader*, pour charger des bases de transactions. Nous donnons également des résultats expérimentaux de notre système sur des bases éparses et denses.

Mots clés : Base de transactions, fonction booléenne, diagramme de décision binaires, densité d'une base de transactions, itemset fréquent.

1 Introduction

L'idée sous-jacente de la fouille de données est d'extraire des connaissances enfouies dans des volumes de données, en croissance continue. De nombreux algorithmes de recherche de connaissances ont été proposés, reposant sur divers modes de représentation de données et de résultats. Ces algorithmes rivalisent en performances car la tâche est coûteuse en temps et en espace mémoire, étant donnée la taille importante des données à exploiter.

Nous nous intéressons à la représentation et à la gestion des bases dites de transactions. À cette fin, nous avons prospecté une nouvelle approche de représentation de bases de transactions en mémoire, basée sur un format condensé des transactions. L'idée est de représenter une table de transactions par une fonction à variables binaires et à valeur entière. Une telle fonction est ensuite représentée et manipulée à l'aide d'une structure de données compacte et canonique appelée diagramme de décision binaire (DDB). À notre connaissance, l'utilisation de cette structure n'a pas encore été envisagée pour représenter ni les bases de transactions, ni les connaissances en fouille de données. Nous présentons un algorithme que nous avons implanté en un système baptisé *BoolLoader*, pour charger des bases de transactions en mémoire, en utilisant les DDB. Nous donnons ici des résultats expérimentaux de notre système sur des bases éparses et denses. L'étude empirique conduit à la conclusion suivante :*BoolLoader* est efficace sur les bases de transactions *denses*, et de façon plus générale, il donne une estimation assez intéressante de la densité d'une base.

[1] À la mémoire de Zahir Maazouzi qui nous a quittés depuis.

Item	Film	Réalisateur
x_1	Harry Potter	C. Columbus
x_2	Star Wars II	G. Lucas
x_3	Attrape moi si tu peux	S. Spielberg
x_4	Un homme d'exception	R. Howard

$\mathcal{D}$	
Tid	**Transaction**
1	x_1, x_2
2	x_1, x_2, x_4
3	x_1, x_2
4	x_3, x_4
5	x_1, x_2
6	x_3, x_4
7	x_1, x_3, x_4
8	x_1, x_2, x_3
9	x_1, x_2
10	x_1, x_3, x_4
11	x_1, x_2
12	x_1, x_2, x_3
13	x_1, x_2
14	x_1, x_3, x_4
15	x_3, x_4

$\xrightarrow{\wp}$

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	3
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	3
1	1	0	0	6
1	1	0	1	1
1	1	1	0	2
1	1	1	1	0

TAB. 1 – Exemple de BDT et la table de vérité qui lui correspond

2 Bases de transactions

Définition 2.1 (Item et itemset) *Un item x_i est un élément d'un ensemble fini $\mathcal{I} = \{x_1, x_2, ..., x_n\}$. On appelle* **itemset** *tout sous-ensemble non vide d'items de $\mathcal{I}$.*

L'ensemble de tous les itemsets forme un treillis, $(\mathcal{P}(\mathcal{I}), \subseteq)$, ayant pour minimum $\perp = \emptyset$ et pour maximum $\top = \mathcal{I}$.

Définition 2.2 (Transaction) *On appelle* transaction *un couple (tid, X), où tid est un identificateur et $X \subseteq \mathcal{I}$.*

Dans la suite, $\mathcal{T}$ désignera l'ensemble des identificateurs de transactions.

Définition 2.3 (Base de transactions) *Une* **base transactionnelle** *$\mathcal{D}$ est un ensemble de couples distincts formés chacun d'un identificateur de transaction* tid *et de la transaction proprement dite.*

$$\mathcal{D} = \{(y, X_y) \ / y \in \mathcal{T},\ X_y \subseteq \mathcal{I}\}$$

Dans la suite, on notera par BDT une base de transactions.

Définition 2.4 *(Itemset fréquent) La* **fréquence**[2] *d'un itemset X est le nombre de transactions de $\mathcal{D}$ contenant X :* fréquence(X)$= |\{(y, X_y) \in \mathcal{D} / X \subseteq X_y\}|$

[2]Lorsque la fréquence est donnée relativement à la taille de la BDT, on l'appelle alors *support.*

Un itemset est dit fréquent dans une BDT si sa fréquence dépasse un seuil minimum de fréquence fixé a priori. Un itemset est dit fréquent maximal, s'il est fréquent et si tous ses sur-ensembles sont non fréquents.

Exemple 1

Considérons la BDT donnée en Table 1 donnant pour 15 cinéphiles, l'ensemble des films qu'ils ont vus récemment. $\mathcal{D}$ est définie sur l'ensemble des items (films) $\mathcal{I} = \{x_1, x_2, x_3, x_4\}$, $\mathcal{T} = \{1, 2, \cdots, 15\}$. Chaque ligne dans $\mathcal{D}$ donne pour chaque cinéphile identifié par un tid unique un ensemble de films. Par exemple, le cinéphile 1 a vu récemment *Harry Potter* et *Star Wars II*. L'itemset $\{x_1, x_3, x_4\}$ (noté $x_1x_3x_4$ pour simplifier) a une fréquence de 3 (ou encore un support de 20%), et si l'on considère un seuil minimum de fréquence de 3 transactions, on dira qu'il est fréquent.

3 Des bases de transactions aux fonctions vectorielles

Théorème 3.1 Isomorphisme de treillis *Le treillis $(\mathcal{P}(\mathcal{I}), \subseteq)$ où $\mathcal{I}$ est un ensemble de cardinal n est isomorphe au treillis $(\mathbb{B}^n, \leq)$ où $\mathbb{B} = \{0, 1\}$.*

En effet, on peut définir une fonction bijective $\wp$ comme suit :

$$\begin{array}{rcl} \wp : \mathcal{P}(\mathcal{I}) & \longrightarrow & \mathbb{B}^n \\ X & \longmapsto & (b_1, b_2, \cdots, b_n) \text{ tq. } b_i = 1 \text{ ssi } x_i \in X,\ 0 \textit{ sinon} \end{array}$$

Ainsi, une combinaison de n items de $\mathcal{I}$ peut être représentée par un vecteur binaire de n-bits, $(b_1, b_2, \cdots, b_n)$, où chaque bit, $b_k \in \mathbb{B}$, exprime si l'item correspondant est inclus dans la combinaison ou non. On définit fréq$(b_1, b_2, \cdots, b_n)$ comme la fréquence d'apparition de la transaction associée.

Considérons une table de vérité $\mathbb{T}^n = [\overrightarrow{e_1^n}, \ldots, \overrightarrow{e_n^n}]$, où pour chaque indice j, $1 \leq j \leq n$, $\overrightarrow{e_j^n}$ représente le j^{ème} vecteur colonne de $\mathbb{T}^n$. Dans $\mathbb{T}^n$, chaque ligne correspond à une transaction possible. On associe à chaque ligne, la fréquence d'apparition de la transaction correspondante dans $\mathcal{D}$. Il est ainsi possible de représenter une BDT par une table de vérité à n variables, n étant le nombre d'items de la BDT, et une fonction vectorielle donnant pour chaque ligne de $\mathbb{T}^n$ la fréquence de la combinaison d'items correspondante dans $\mathcal{D}$. Étant donné que la structure de la table de vérité est fixée, et que les n items sont ordonnés, la fonction f de taille 2^n est alors suffisante pour exprimer $\mathcal{D}$.

Exemple 2

La BDT $\mathcal{D}$ de la table 1 est représentée par la table de vérité (à droite) suivante : $\mathbb{T}^4 = [\overrightarrow{e_1^4}, \ldots, \overrightarrow{e_4^4}]$ où $\overrightarrow{e_1^4}$ = 0000 0000 1111 1111, $\overrightarrow{e_2^4}$ = 0000 1111 0000 1111, $\overrightarrow{e_3^4}$ = 0011 0011 0011 0011, $\overrightarrow{e_4^4}$ = 0101 0101 0101 0101.
et ayant pour fonction vectorielle $\overrightarrow{f}$ = 0003 0000 0003 6120.
Par d'exemple, la transaction $\{x_1, x_2, x_3\}$ existe 2 fois dans $\mathcal{D}$, elle est représentée par la ligne (1110) de la table de vérité et la valeur de f pour cette entrée est 2. De façon similaire la transaction $\{x_1, x_2\}$ est représentée par l'entrée (1100), la valeur de la fonction est égale à 6 correspondant au nombre de transactions $\{x_1, x_2\}$ dans $\mathcal{D}$.

Étant donné que f représente une forme condensée de $\mathcal{D}$, peut-on alors, en cas de besoin, charger f à la place de $\mathcal{D}$ en mémoire ? De plus, sachant que la taille de f peut être très grande, comment pourrions nous dans ce cas représenter et manipuler efficacement f ? Par exemple, si $\mathcal{D}$ est définie sur 100 items, nous aurons besoin d'une fonction vectorielle de taille égale à 2^{100}, soit plus de 10^{30} entiers naturels ! Les Diagrammes de Décision Binaires (notés DDB dans la suite), proposés par Lee [Lee, 1959] et rendus populaires par Akers [Akers, 1978] et Moret [Moret, 1982], sont des structures compactes permettant de représenter et de manipuler efficacement de telles fonctions.

4 Qu'est ce qu'un diagramme de décision binaire ?

Un DDB est un graphe dirigé acyclique ayant deux nœuds terminaux 0 et 1. Chaque nœud non terminal possède un indice identifiant la variable correspondante de la fonction. Tous les nœuds d'un même niveau ont le même indice. Un nœud non terminal possède aussi deux arêtes sortantes : une continue indique que la variable du nœud est fixée à 1, l'autre discontinue indiquant qu'elle est fixée à 0.
Un DDB représente une fonction de façon compacte et canonique grâce aux propriétés suivantes (figure 1) :

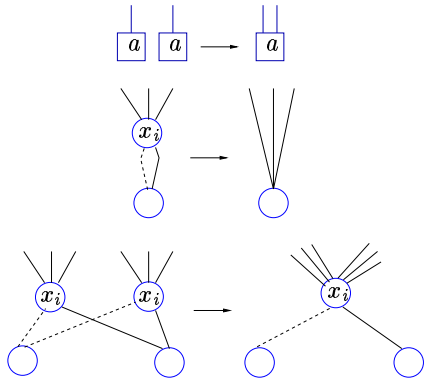

FIG. 1 – Règles de réduction d'un DDB

1. l'ordre sur les variables est fixé : $x_1 \prec x_2 \prec \ldots \prec x_n$
2. chaque nœud non terminal a un indice inférieur à tous les indices de ses nœuds descendants,
3. tous les nœuds redondants, *i.e.* dont les deux arêtes pointent vers un même nœud sont éliminés,
4. tous les sous-graphes isomorphes sont partagés.

Les diagrammes de décision algébriques introduits dans [R.I. Bahar *et al.*, 1993] sont des DDB qui gèrent des nœuds terminaux ayant des valeurs entières positives ou nulles. Ce type de DDB permet de gérer des fonctions à variables booléennes et à valeur entière, comme c'est le cas des fonctions associées aux bases de transactions définies au § 3.
Les opérations sur les DDB ont été définies dans [Bryant, 1986]. L'opération *application* permet d'effectuer des opérations basiques entre DDB, telles que le ET ($\wedge$) et le OU ($\vee$). Nous verrons dans la suite leur utilité dans le cadre de notre travail.

Exemple 3
La figure 3(o) donne le DDB associé à la fonction f de la table 1. Le chemin le plus à droite allant de la racine à la feuille 6 exprime qu'il y a 6 cinéphiles ayant vu Harry Potter (x_1) et Stars Wars II (x_2) mais n'ayant pas vu le reste des films. Le chemin le plus à gauche menant vers 0 exprime qu'aucun cinéphile n'a vu Stars Wars II (x_2) sans avoir vu Harry Potter (x_1).

5 Des bases de transactions aux DDB

```
Algorithme BDT_2_DDB
 Entrée :  Base de transactions 𝒟
 Sortie :  DDB_𝒟
 1.  DDB_𝒟=NULL
 2.  Pour toute transaction t ∈ 𝒟
 3.  faire
 4.        DDB_t=NULL
 5.        Pour i= 1 à n
 6.        faire
 7.           si x_i ∈ t alors DDB_t=DDB_t ∧ DDB_{x_i}
 8.           sinon DDB_t=DDB_t ∧ DDB_{¬x_i}
 9.        fin pour
10.        DDB_𝒟=DDB_𝒟 ∨ DDB_t
11. fin pour
```

La construction d'un DDB représentant une BDT se fait en parcourant toutes les transactions. Un DDB est construit pour chaque transaction et est 'rajouté' au DDB construit jusqu'à présent en effectuant l'opération $\vee$. L'algorithme général *BDT_2_DDB* de transformation d'une BDT en DDB est donné ci-dessus. Pour obtenir un DDB aussi compact que possible, en plus des règles de réduction des DDB, on ordonne les items dans les transactions par ordre décroissant de fréquence : que les items les plus fréquents se trouveront aussi proches de la racine et seront donc partagés par un maximum de transactions. Notons enfin que la fonction f, à proprement parler, n'est jamais calculée, sa taille étant très grande, en effet, on passe directement de la base de transactions au DDB. De façon plus générale, la construction d'un DDB associé à un BDT s'effectue en considérant les disjonctions des transactions qui sont elle mêmes des conjonctions d'items ou de négation d'items.

Lemme 5.1 *L'algorithme* BDT_2_DDB *associe à une base de transactions $\mathcal{D}$ un diagramme de décision représentant toutes ses transactions et seulement ses transactions.*

Exemple 4

La figure 2 montre les étapes de construction du DDB de la première transaction de la BDT, $x_1 \wedge x_2$, que l'on peut écrire $x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4$ puisque les items x_3 et x_4 ne sont pas présents dans la transaction. La figure 3 représente les étapes de construction du DDB correspondant à la base de transactions de la table 1. La figure 3(a) représente le DDB de la 1ère transaction, la (b) représente le DDB des deux premières transactions et ainsi de suite, la table 1 est enfin représentée par (o).

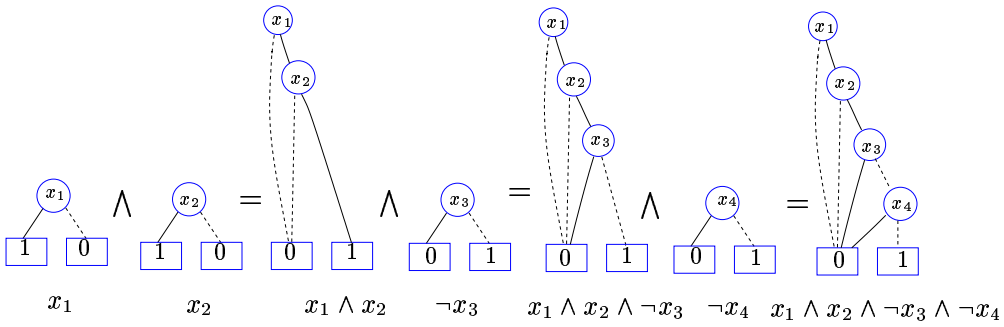

FIG. 2 – Construction du DDB de la transaction 1

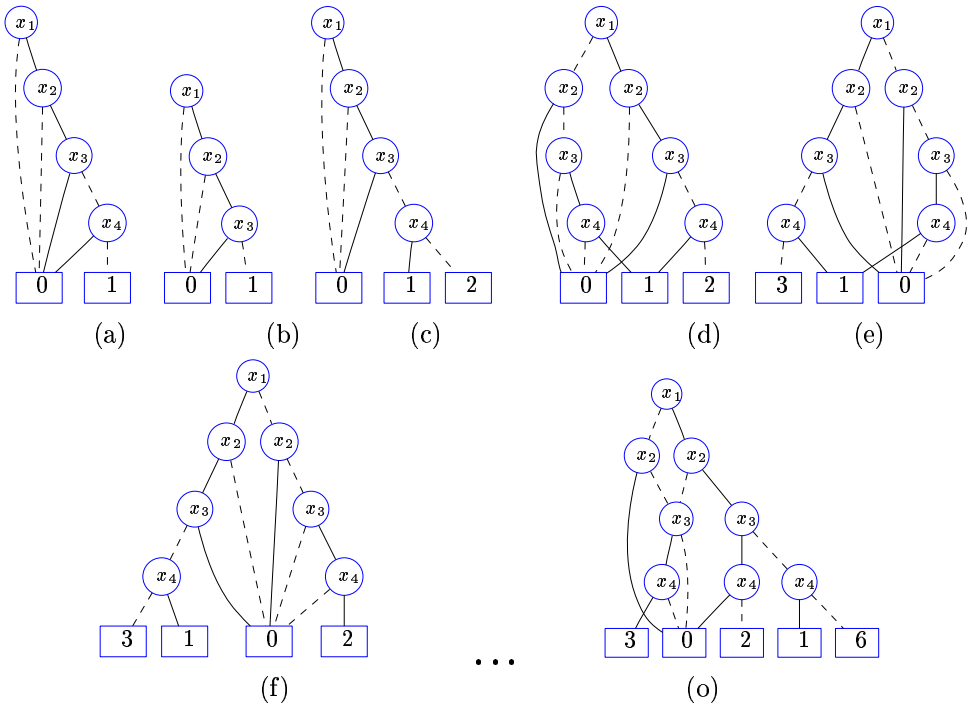

FIG. 3 – Construction du DDB de la table 1

6 Implémentation et expérimentations

Nous avons développé en langage C, *BoolLoader*, un prototype de chargement des bases de transactions, basé sur l'algorithme *BDT_2_DDB*. Nous avons utilisé comme principale structure de données, les DDB ayant des feuilles entières positives ou nulles ainsi que les DDB partagés (Shared DDB [Minato, 1990]) pour optimiser l'utilisation de la mémoire. Pour cela, nous avons choisi la librairie CUDD[3]. Elle peut gérer des DDB avec un nombre de nœuds pouvant atteindre 2^{28}, soit plus de 250 millions de nœuds. Les nœuds sont de taille 16 octets, soit parmi les tailles les plus petites des librairies existantes. Le nombre de variables maximales gérées par CUDD est de 2^{16} soit 65 536 variables. La librairie CUDD utilise un système de *cache* bien utile lors des calculs des opérations entre DDB. De plus, un système de *ramasse miettes* est utilisé, il supprime de la mémoire les nœuds des DDB qui ne sont plus utilisés. Nous donnons dans ce qui suit les résultats expérimentaux obtenus avec *BoolLoader* sur des bases transactionnelles synthétiques et réelles. Les informations concernant ces bases sont données dans le tableau 2. $\mathcal{D}$ désigne le nom de la base de transactions, N le nombre d'items, T la taille moyenne des transactions, D le nombre de transactions et enfin la *taille* du fichier sur disque. Les expérimentations ont été effectuées sur un PC Pentium cadencé à 500 Mhz, doté de 256 méga octets de mémoire principale et fonctionnant sous Linux.

Les expérimentations ont montré que les performances de *BoolLoader* sont très bonnes pour le chargement des BDT denses telles que *mushroom* et *connect*. Dans ce cas, on a obtenu des taux de compression très intéressants donnés en table 3. Pour *chess* le taux est moins élevé. Cela provient très probablement du fait que les transactions dans chess se partagent des itemsets longs et courts. Ainsi certains chemins du DDB de chess sont bien partagés et d'autres pas du tout.

La figure 4 donne les résultats de chargement de la BDT synthétique T10I4D10K en utilisant *BoolLoader*. Si le temps de chargement en mémoire est relativement petit (figure 4(a)), nous avons été surpris par le nombre de nœuds du DDB correspondant à T10I4D10K (figure 4(b)). La taille du DDB (figure 4(c)) est nettement supérieure à la taille de la BDT (figure 4(d)). Nous avons alors effectué d'autres tests avec d'autres BDT synthétiques telles que T10I4D100K et T20I6D10K et nous avons constaté un comportement de *BoolLoader* similaire à celui de T10I4D10K. La raison est que ces BDT sont éparses, c'est même le pire des cas pour notre chargeur. Les itemsets maximaux dans de telles BDT sont très courts. En moyenne, pour T10I4D10K avec N=1000 la taille moyenne des itemsets est de 4, peu de transactions sont identiques, et peu se partagent des itemsets longs.

Afin de réduire le nombre de nœuds, on a utilisé une heuristique basée sur la fréquence : on a réduit dans une étape de pré-traitement le nombre d'items en ôtant les items non fréquents par rapport à un seuil donné. On voit ainsi sur la figure 4 les courbes décroissantes du nombre de nœuds, ainsi que du temps de chargement lorsque l'on augmente le support minimum. Afin de confirmer que *BoolLoader* est meilleur pour les BDT denses comparativement aux BDT éparses, nous avons généré des BDT synthétiques pour T10I4D10K avec un nombre d'items N de 50, 100, 500 et 1000. La figure 4 montre que le chargement est plus efficace pour N petit mais aussi que

[3] `http://www.bdd-portal.org/cud.html`, `http://vlsi.colorado.edu/~fabio/CUDD/`

$\mathcal{D}$	N	T	D	taille
T10I4D10K	50..1000	10	10K	500 Ko
Mushroom	120	23	8 124	663 Ko
Connect	130	43	67 557	9 Mo
Chess	76	37	3 196	375 Ko

TAB. 2 – Bases de transactions utilisées dans les tests

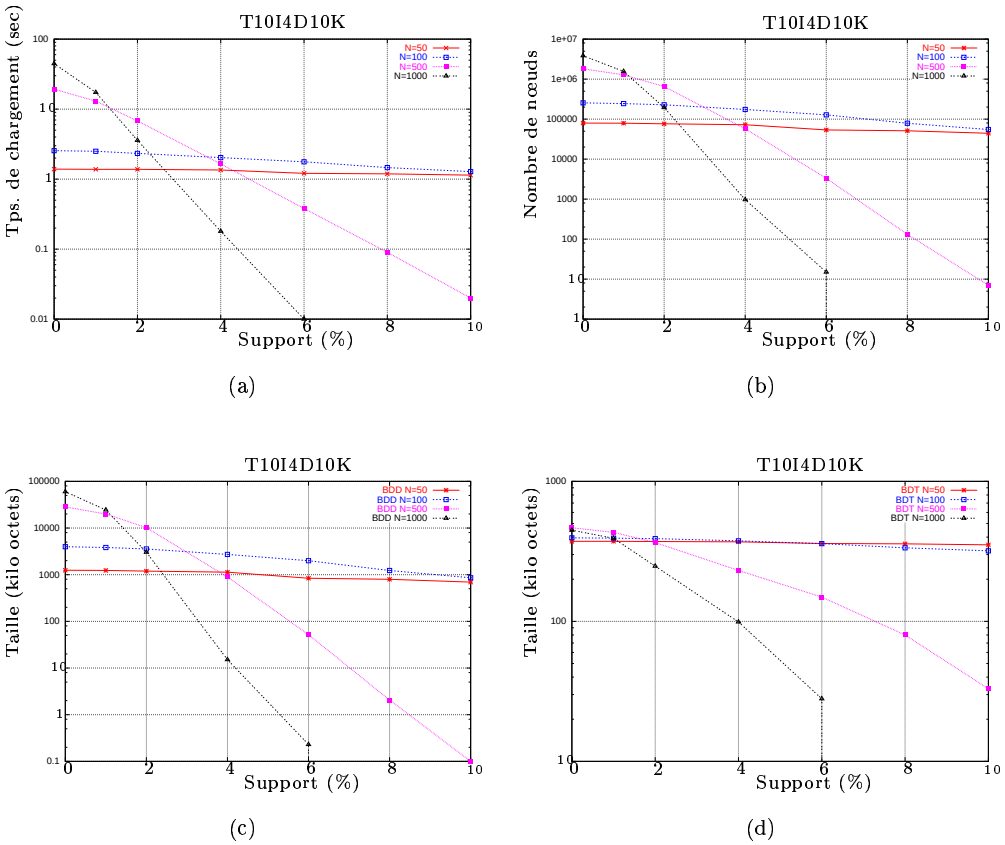

FIG. 4 – Résultats expérimentaux sur T10I4D10K

$\mathcal{D}$	|BDT|	#nœuds	|DDB|	Temps(s)	Taux de compression
Mushroom	663 Kb	3223	50 Kb	3	92.5%
Connect	9 Mb	171 662	2.6 Mb	47	72%
Chess	375 Kb	18 553	289 Kb	1	23%

TAB. 3 – Résultats expérimentaux sur des bases denses

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	3
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	3
1	1	0	0	6
1	1	0	1	1
1	1	1	0	2
1	1	1	1	0

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	3
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	3
1	1	0	0	6
1	1	0	1	1
1	1	1	0	2
1	1	1	1	0

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	3
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	3
1	1	0	0	6
1	1	0	1	1
1	1	1	0	2
1	1	1	1	0

TAB. 4 – Lignes de tables de vérité associées à x_2, x_2x_4, $x_2x_3\neg x_4$ (de g. à d. resp.)

l'heuristique adoptée plus haut n'est pas avantageuse lorsque la BDT est dense.

7 Application : calcul de la fréquence d'un itemset

Étant donnée une BDT, $\mathcal{D}$ la fonction f correspondante [4] et un itemset X de taille k, comment pouvons nous calculer la fréquence de X dans $\mathcal{D}$? Pour cela, définissons d'abord l'opérateur *Prod*, ($i_1 \cdots i_k$ sont les indices des variables présentes dans X) :

$$\begin{array}{llll} Prod : & \mathcal{P}(\mathcal{I}) & \longrightarrow & \mathbb{N}^{2^n} \\ & X & \longmapsto & f_X = f \times e^n_{i_1} \times \cdots \times e^n_{i_k} \end{array}$$

D'une certaine manière, le produit met l'accent sur certaines positions de la fonction vectorielle en mettant à zéro le reste des positions, ce qui permet de garder les transactions concernées par un itemset X. Notons que le vecteur f_X associé à X peut être représenté par un sous-DDB du DDB de f. *Prod* est accompli en utilisant l'opération *application* entre DDB. Notons enfin que pour pour calculer fréquence(X), il suffit de faire la somme du vecteur f_X qui est associé à X.

Exemple 5
Considérons les itemsets : x_2, x_2x_4, $x_2x_3\neg x_4$. Intuitivement, le dernier itemset exprime qu'on s'intéresse aux spectateurs ayant vu *Star Wars II* et *Attrape moi si tu peux* mais n'ayant pas vu *Un homme d'exception*, de plus ils peuvent avoir vu *Harry Potter* ou non. On verra à travers cet itemset l'extension de *Prod* aux itemsets avec négation. La table 4 donne les vecteurs correspondant aux itemsets, la figure 5 donne les DDB associés.

$$\begin{array}{lll} f_{x_2} & = & f \times e_2 = (0003\ 0000\ 0003\ 6120) \times (0000\ 1111\ 0000\ 1111) \\ & = & 0000\ 0000\ 0000\ 6120 \\ f_{x_2x_4} & = & (f \times e_2) \times e_4 = f_{x_2} \times (0101\ 0101\ 0101\ 0101) = 0000\ 0000\ 0000\ 0100 \\ f_{x_2x_3\neg x_4} & = & (f \times e_2) \times e_3 \times \neg e_4 \\ & = & f_{x_2} \times (0011\ 0011\ 0011\ 0011) \times (1010\ 1010\ 1010\ 1010) = 0000\ 0000\ 0000\ 0020 \end{array}$$

[4] pour simplifier l'écriture on abandonne les flèches sur les vecteurs

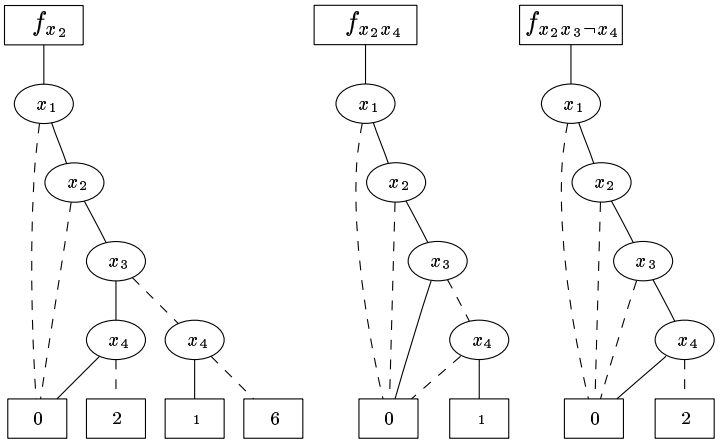

FIG. 5 – DDB associés à x_2, x_2x_4, $x_2x_3\neg x_4$

8 Formats des BDT : état de l'art

Dans la littérature, on peut distinguer trois principales tendances de représentation des bases de transactions : le **format horizontal** utilisé dans l'algorithme *Apriori* [Agrawal et Srikant, 1994] et quelques unes de ses variantes, c'est le format pionnier de représentation des BDT. Ces dernières sont vues horizontalement comme des ensembles de tids. Le **format vertical** consiste à voir la BDT verticalement, à chaque item est associé un ensemble de tids, *tidset*. Ce format est utilisé, par exemple, dans *Eclat* [Zaki *et al.*, 1997]. Un format récent optimisant les tidsets est le format *Diffset* proposé dans [Zaki et Gouda, 2003]. L'idée est de garder les différences entre les tidsets pour optimiser le calcul du support. Les *vecteurs de bits* utilisés dans *Mafia* [Burdick *et al.*, 2001] ainsi que dans *Viper* [Shenoy *et al.*, 2000] représentent aussi les BDT verticalement mais plutôt comme des vecteurs de bits et non des tidsets. Enfin, le **format condensé sous forme de graphe** tel que *Fp-tree* [Han *et al.*, 2000] représente une BDT par un arbre préfixé représentant les transactions. Cette structure est utilisée dans l'algorithme *Fp-growth* [Han *et al.*, 2000] et *Closet* [Pei *et al.*, 2000]. Selon son algorithme de construction, le *FP-tree* se base principalement sur un partage des préfixes de transactions et non des suffixes comme c'est le cas pour les DDB, qui rentrent donc naturellement dans la troisième tendance de représentation. Au vu des expérimentations présentées dans [Zaki et Gouda, 2003], où les tailles de quelques BDT denses et éparses sont comparées dans différents formats, et les expérimentations que nous avons menées, il nous semble que les DDB représentent une structure de données prometteuse et très adaptée aux données denses.

9 Conclusion et perspectives

Nous avons proposé et étudié une approche nouvelle pour le chargement, quand c'est possible, de bases de transactions en mémoire à l'aide de diagrammes de décision. L'étude empirique a permis de montrer que cette représentation condensée donnait de bons résultats de chargement et de compression sur les bases de données denses comparativement aux bases de données éparses. Dans ce dernier cas, une heuristique d'élagage des itemsets de taille 1 permet de réduire le nombre de nœuds dans les DDB correspondants. En plus du chargement efficace des bases denses, ce qui est le cas dans les applications réelles, *BoolLoader* permet de donner une estimation de la densité de la base de données. On pourrait alors l'utiliser à des fins de prétraitement nous permettant d'avoir les caractéristiques en terme de densité de la base de transactions sur laquelle nous souhaitons effectuer des traitements de fouille de données. Un deuxième point concerne la sauvegarde des bases de transactions. En effet, le DDB généré par *BoolLoader* pourrait être sauvé sur disque dans le cas de bases denses par exemple. Un autre point important est que la densité des bases de transactions dépend de la variété et du nombre des motifs maximaux trouvés, et cette notion mériterait d'être approfondie. Le recours au partitionnement de la base de transactions peut s'avérer indispensable, car même si la structure de données employée est très compacte, on peut se retrouver face à des bases de transactions volumineuses, dont les DDB ne peuvent être chargés en mémoire.

Dans le domaine d'extraction des itemsets fréquents, beaucoup d'algorithmes sont inefficaces ou échouent sur les BDT denses, considérées comme des BDT difficiles. C'est le cas par exemple de l'algorithme Apriori. Comme perspectives de ce travail, nous projetons d'une part de faire une étude empirique comparative entre tous les formats et d'autre part d'utiliser cette structure dans la tâche d'extraction des motifs fréquents. Pour ce dernier point, nous avons déjà eu une première expérience [Salleb *et al.*, 2002] où nous avons proposé un algorithme montrant la faisabilité de l'extraction des maximaux à partir d'un DDB et nous souhaitons tester d'autres algorithmes de recherche sur les BDD. Nous pensons enfin utiliser les DDB comme structure de données pour représenter efficacement cette fois ci les connaissances apprises.

Remerciements : Nous tenons à remercier les relecteurs dont les remarques nous ont été d'une grande utilité pour l'amélioration de cet article.

Références

[Agrawal et Srikant, 1994] Rakesh Agrawal et Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, et Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 1994.

[Akers, 1978] S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27 :509–516, 1978.

[Bryant, 1986] R.E. Bryant. Graph-based algorithms for boolean functions manipulation. *IEEE Trans. on Computers*, C-35(8) :677–691, August 1986.

[Burdick *et al.*, 2001] D. Burdick, M. Calimlim, et J. Gehrke. Mafia : A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 443–452, Heidelberg, Germany, April 2001.

[Han *et al.*, 2000] J. Han, J. Pei, et Y. Yin. Mining frequent patterns without candidate generation. *In ACM-SIGMOD Int. Conf. on Management of Data*, pages 1–12, 2000.

[Lee, 1959] C.Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell Systems Technical Journal*, 38 :985–999, July 1959.

[Minato, 1990] S. Minato. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proc. 27th Design Automation Conference*, pages 52–57, June 1990.

[Moret, 1982] B.M. Moret. Decision trees and diagrams. *ACM Computing Surveys*, 14(4) :593–623, 1982.

[Pei *et al.*, 2000] Jian Pei, Jiawei Han, et Runying Mao. CLOSET : An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[R.I. Bahar *et al.*, 1993] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, et F. Somenzi. Algebraic decision diagrams and their applications. In *IEEE /ACM International Conference on CAD*, pages 188–191, Santa Clara, California, 1993. IEEE Computer Society Press.

[Salleb *et al.*, 2002] A. Salleb, Z. Maazouzi, et C. Vrain. Mining maximal frequent itemsets by a boolean approach. In IOS Press Amsterdam F. van Harmelen, editor, *ECAI*, pages 385–389, Lyon, France, July 21-26 2002. Proceedings of the 15th European Conference on Artificial Intelligence.

[Shenoy *et al.*, 2000] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, , et D. Shah. Turbo-charging vertical mining of large databases. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, May 2000.

[Zaki *et al.*, 1997] M. J. Zaki, S. Parthasarathy, M. Ogihara, et W. Li. New algorithms for fast discovery of association rules. In David Heckerman, Heikki Mannila, Daryl Pregibon, Ramasamy Uthurusamy, et Menlo Park, editors, *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 283–296. AAAI Press, 1997.

[Zaki et Gouda, 2003] M. J. Zaki et K. Gouda. Fast vertical mining using diffsets. In *9th International Conference on Knowledge Discovery and Data Mining*, Washington, DC, August 2003.

Summary

We address the problem of representing and loading transactional datasets relying on a compressed representation of all the transactions of that dataset. We present an efficient algorithm, named *BoolLoader*, to load transactional datasets and we give an experimental evaluation on both sparse and dense datasets. It shows that *BoolLoader* is efficient for loading dense datasets and gives a useful tool to evaluate the density of any dataset. **Keywords :** Transactional datasets, boolean functions, decision diagrams, dense datasets, frequent itemsets.