Estimation of the Density of Datasets with Decision Diagrams

Ansaf Salleb[1] Christel Vrain[2]

[1] IRISA-INRIA Campus Universitaire de Beaulieu 35042 Rennes Cedex - France
Ansaf.Salleb@irisa.fr**

[2] LIFO, Université d'Orléans, B.P. 6759, 45067 Orléans Cedex 2 - France
Christel.Vrain@lifo.univ-orleans.fr

We dedicate this paper to our colleague Zahir Maazouzi who participated to that work.

Abstract. We address the problem of loading transactional datasets into main memory and estimating the density of such datasets. We propose BoolLoader, an algorithm dedicated to these tasks; it relies on a compressed representation of all the transactions of the dataset. For sake of efficiency, we have chosen Decision Diagrams as the main data structure to the representation of datasets into memory. We give an experimental evaluation of our algorithm on both dense and sparse datasets. Experiments have shown that BoolLoader is efficient for loading some dense datasets and gives a partial answer about the nature of the dataset before time-consuming patterns extraction tasks.
Keywords: Transactional dataset, boolean function, decision diagram, density

1 Introduction

Many works have addressed the problem of efficiently mining frequent itemsets and frequent association rules in transactional databases (for instance [1, 3, 8, 16, 21]). One of the key problems is to find an appropriate data structure for loading the transactional database into main memory, and the efficiency depends on the nature of the database, e.g., density / sparseness. Indeed, choosing the right algorithm is difficult without *a priori* information on the database.

In this paper, we develop two points. First, we study the interest of *Binary Decision Diagrams (BDDs)* as a data structure for representing and loading transactional datasets. Then we introduce a coefficient, called the *sparseness coefficient* and we experimentally show that it could be an interesting measure for evaluating the density of a database. In our framework, a dataset is viewed as a vectorial function, thus allowing, when possible, to load only this vectorial function in memory by means of a BDD. Such a structure has already been successfully used for representing boolean functions in various applications of Computer Science such as very large scale integration systems. As far as we know, it has not yet been studied in the field of mining transactional databases. For the time being, four tendencies for representing and handling transactional datasets can be distinguished :

** This work was done while the first author was a PhD student at LIFO, University of Orléans.

• **Horizontal format**: Used in the *Apriori* algorithm [1], this format is considered as the classical representation. The dataset is seen as a succession of transactions, each one identified by an identifier *tid*. This format is also used for mining maximal frequent itemsets in algorithms such as *MaxMiner* [3].
• **Vertical format**: It is used by *Eclat* [21] and *Partition* [16]. It consists in representing the dataset vertically by giving to each item its *tidset*, *i.e.* the set of transactions containing this item. Another recent vertical format named *Diffset* has been proposed in [20]. It consists in keeping only track of differences between tidsets.
• **Bitvectors**: Bitvectors are used in the algorithm *Mafia* [5] and *Viper* [17]. This format consists in representing data as bitvectors compressed using a strategy called *Run Length Encoding*.
• **Fp-tree**: This data structure is an extended prefix-tree structure for storing compressed and crucial information about frequent patterns. This data structure has been used in the *Fp-growth* [8] algorithm for mining frequent patterns.
In this paper, we are interested in developing a new representation and we propose an algorithm, called BoolLoader, to load transactional datasets. We also give an experimental evaluation on both sparse and dense datasets. It shows that BoolLoader is particularly efficient for loading dense datasets which are considered as challenging datasets in mining frequent itemsets. Moreover, comparing the size of the BDD and the initial size of the database gives an interesting measure for evaluating its density, and can be very important information on the nature of the dataset before working on it.

The remainder of this paper is organized as follows: In § 2 we give some basic definitions concerning itemsets and transactional datasets. We then show in § 3 how to represent a dataset by a vectorial function. § 4 is devoted to the BDD data structure. In § 5, we propose an algorithm for moving from a given dataset to a BDD. Details on implementation and experimental tests are given in § 6. We finally conclude in § 7.

2 Transactional Datasets and Frequent Itemsets

This section recall some definitions concerning the frequent itemset mining task. An *item* is an element of a finite set $\mathcal{I} = \{x_1, ..., x_n\}$. A subset of $\mathcal{I}$ is called an *itemset*. The set of all possible itemsets ordered by set inclusion forms a lattice $(\mathcal{P}(\mathcal{I}), \subseteq)$. A *transaction* is a subset of $\mathcal{I}$, identified by a unique transaction identifier *tid*. $\mathcal{T}$ denotes the set of all transaction identifiers. A transactional database is a finite set of pairs (y, X_y) where y is a transaction identifier and X_y is an itemset; in the following, it is denoted by *BDT*. The *frequency* of an itemset X in a BDT $\mathcal{D}$ is the number of transactions in $\mathcal{D}$ containing X. An itemset X is said to be frequent in $\mathcal{D}$ when its frequency is greater than a given threshold.

Example 1. Let us consider the BDT given in Table 1 and let us suppose that it stores movies recently seen by 15 spectators. The dataset $\mathcal{D}$ is defined on the set of items (movies) $\mathcal{I} = \{x_1, x_2, x_3, x_4\}$. The set of tids is given by $\mathcal{T} = \{1, 2, \cdots, 15\}$. Each line in $\mathcal{D}$ associates a set of movies to the spectator identified by the corresponding tid. For instance, the spectator 1 has recently seen *Harry Potter* and *Star Wars II*. The itemset $\{x_1, x_2\}$, written x_1x_2 for sake of simplicity, is frequent relatively to the threshold 2 since it appears 9 times in $\mathcal{D}$.

Item	Movie	Producer
x_1	Harry Potter	C. Columbus
x_2	Star Wars II	G. Lucas
x_3	Catch me if you can	S. Spielberg
x_4	A Beautiful Mind	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

$\xrightarrow{\wp}$

e_1	e_2	e_3	e_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

Table 1. A transactional dataset and its corresponding truth table

3 From Transactional Datasets to Vectorial Functions

Our framework relies on the Stone's representation theorem for Boolean algebras [18]:

Theorem 1. Lattice isomorphism *The lattice $(\mathcal{P}(\mathcal{I}), \subseteq)$ where $\mathcal{I}$ is a set of n items is isomorphic to the lattice $(\mathbb{B}^n, \leq)$ where $\mathbb{B}$=$\{0,1\}$ and $(b_1, \ldots, b_n) \leq (b'_1, \ldots, b'_n)$ when for all i, $b_i \leq b'_i$.*

The bijective function $\wp$ is defined by $\wp(X) = (b_1, b_2, \cdots, b_n)$ where $b_i = 1$ if $x_i \in X$, 0 $otherwise$. Thus, each bit expresses whether the corresponding item x_i is included in that combination or not. Let us consider a truth table $\mathbb{T}^n = [e_1, \ldots, e_n]$, where for each index j, $1 \leq j \leq n$, e_j is a 2^n-bits vector representing the jth vector of $\mathbb{T}^n$. In $\mathbb{T}^n$ each line corresponds to a possible combination of $b_1, \ldots, b_n$, thus to an itemset. We can associate to this truth table a vectorial function f, which gives for each line of the truth table (a combination of items), the number of times the transaction corresponding to that itemset appears in $\mathcal{D}$. Since the structure of the truth table is fixed when n is fixed and when the variables are ordered, the function f is then sufficient to express the entire set of transactions of $\mathcal{D}$.

Example 2. The BDT $\mathcal{D}$ of table 1 is represented by $\mathbb{T}^4 = [e_1, \ldots, e_4]$ where:

e_1 `= 0000 0000 1111 1111` e_2 `= 0000 1111 0000 1111`

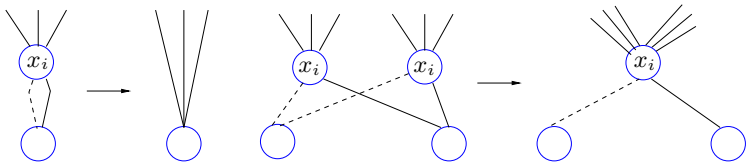
e_3 `= 0011 0011 0011 0011` e_4 `= 0101 0101 0101 0101`

with the output function f `= 0003 0000 0003 6120`.

For instance, the transaction $\{x_1, x_2, x_3\}$ exists twice in $\mathcal{D}$, it is then represented by the 15^{th} line (1110) in the truth table and the value of f is equal to 2. In the same way, the transaction $\{x_1, x_2, x_3, x_4\}$ does not exist in $\mathcal{D}$, it will be represented by the last line (1111) in the truth table with 0 as output function.

The vector f represents a new form of the dataset. It is then interesting to study whether it is possible to load it into memory instead of loading the dataset itself. This seems very difficult since the size of f may be very large. For instance, for a dataset defined on 100 items, the size of the corresponding vectorial function is equal to 2^{100}, so greater than 10^{30} unsigned integers. But, a compact representation, called BDD, has been introduced by Lee [9] and Akers [2]. Moreover, we show in Section 5 that it is possible to build the BDD directly from the dataset without computing f.

4 Binary and Algebraic Decision Diagrams



(a) Node elimination

(b) Node sharing

Fig. 1. Reduction rules

A Binary Decision Diagram (BDD) is a graph-based representation of boolean functions. It is a directed acyclic graph with 2 terminal nodes 1 and 0. Each non-terminal node has an index to identify a variable of the boolean function, and has two outgoing edges; the dashed one means that the variable is fixed to 0 whereas the other one means that the variable is fixed to 1. A BDD represents a disjunctive normal form of a boolean function: each path from the root of a BDD till a leave indexed by number 1 gives a conjunction of literals (where a literal is either a variable or the negation of a variable) that is true for that boolean function. Given a boolean function, it is possible to represent it by a canonical graph, using the following rules [11] (Figure 1):

1. Choose an order on variables: $x_1 \prec x_2 \prec \ldots \prec x_n$; variables appear in this order in all the paths of the graph and no variable appears more than once in a path.
2. Eliminate all the redundant nodes whose two edges point to the same node.
3. Share all equivalent subgraphs.

Operations (AND ($\wedge$), OR ($\vee$), etc.) on BDDs have been defined in [4]. For example, the BDD of the expression $x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4$ given in Figure 2 is obtained by first generating the trivial BDDs of x_1, x_2, $\neg x_3$ and $\neg x_4$, and then by computing the AND operation between these basic BDDs. In our case, we have to handle vectorial functions from $\mathbb{B}^n$ to $\mathbb{N}$. We use an extension of BDD, called Algebraic Decision Diagrams (ADD) [6, 13] that handles such functions: in ADDs, leaves are indexed by integers. In the following, we still use the term BDD instead of ADD, since it is more commonly used.

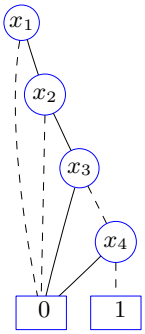


Fig. 2. BDD of $x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg x_4$

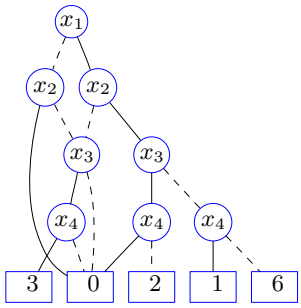


Fig. 3. BDD corresponding to the vector f

Example 3. Figure 3 gives the BDD of the function f of Table 1. For instance, the rightmost path expresses that there are 6 spectators who have seen *Harry Potter* (x_1) and *Stars Wars* (x_2) but who have not seen the other movies (x_3, x_4). The leftmost path expresses that no spectator has seen *Stars Wars* without having seen *Harry Potter*.

5 From Datasets to Decision Diagrams

5.1 Building a Binary Decision Diagram

The construction of a BDD representing a dataset is done by scanning only once the dataset. For each transaction, a BDD is constructed and added to the final BDD using the operation $\vee$ between BDDs. Although not shown in the algorithm, reduction rules (eliminating redundant nodes and sharing equivalent subgraphs) (Figure 1) are used during the construction process in order to get a compact BDD. Let us notice that **the function f is never computed**; in fact, we transform directly a transactional dataset into its corresponding BDD. The construction of a BDD associated to a BDT is given by the algorithm *BDT2BDD*.

```
  Algorithm BDT2BDD
Input : a dataset 𝒟
Output : a decision diagram BDD_𝒟

1. BDD_𝒟=NULL
2. For each transaction t ∈ 𝒟 do
3.     BDD_t=NULL
4.     For i=1 to n do
5.        If x_i ∈ t then BDD_t=BDD_t ∧ BDD_{x_i}
6.        else BDD_t=BDD_t∧ BDD_{¬x_i}
7.     BDD_𝒟=BDD_𝒟 ∨ BDD_t
```

Example 4. Figure 4 represents the construction steps of the BDD of Table 1, considering the transactions one by one. Figure 4(a) represents the BDD of transaction 1, (b) shows the BDD of the two first transactions, (c) the three first transactions and so on. Finally, the BDD in (d) represents all the transactions of the database. For more details about the operations between BDDs representing transactions see [15].

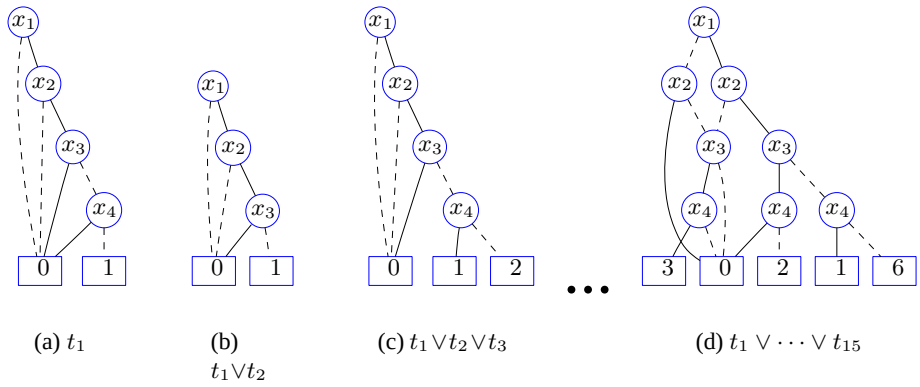


Fig. 4. Example of construction of a reduced BDD of $\mathcal{D}$

5.2 A measure of sparseness

We introduce a measure, called the Sparseness coefficient, defined as follows:

$$\text{Sparseness coefficient} = \frac{\#\text{nodes in BDD}}{\#\text{transactions} \times \#\text{items}}\%$$

It compares the size of the BDD with the size of the database, expressed by the two dimensions that are: the number of items and the number of transactions. It gives an evaluation of the sparseness of the database; a low coefficient would be an indication of a dense database.

6 Implementation and experimental results

We developed a prototype, called BoolLoader to load transactional datasets. It has been developed in C and it uses ADDs as the main data structure for representing datasets and shared BDDs [10] to optimize memory. Our implementation relies on the CUDD[3] library. This free library can manage BDDs with a number of nodes up to 2^{28}, *i.e.*, more than 250 million nodes! The nodes have a size of 16 bytes, one of the smallest sizes of the existing libraries. The maximal number of variables managed by CUDD is equal to 2^{16}, i.e., $65\,536$ variables. The aim of the experiments is twofold: first, to test whether that data structure is suitable for loading transactional datasets, second, to study the sparseness coefficient introduced in Section 5 as an estimation of the density of the database. Experiments have been performed on a PC Pentium 4, 2.66 GHz processor with 512 Mb of main memory, running under Linux Mandrake 9.2. We have tested BoolLoader on real and artificial datasets (Table 2). Artificial datasets have been generated using the *IBM generator*[4]. Concerning real dense datasets, we have experimented

[3] `http://www.bdd-portal.org/cud.html`
[4] `http://www.almaden.ibm.com/cs/quest/syndata.html`

Database	#items	T	#transactions	#nodes	Time(s)	Spars. Coef.(%)
Mushroom	120	23	8 124	3225	1.47	0.334
Connect	130	43	67 557	171 662	26.39	1.970
Chess	76	37	3 196	18 553	0.54	7.741
T10I4D10K	50	10	9 823	102 110	1.56	20.79
	100	10	9 824	316 511	2.94	32.21
	500	10	9 853	2 557 122	25.83	51.90
	1000	10	9 820	5 480 596	59.57	58.81
T10I4D100K	50	10	98 273	479 382	2.32	9.75
	100	10	98 394	1 985 437	55.41	20.17
	150	10	98 387	4 131 643	93.04	28.18
T10I8D100K	50	10	84 824	442 671	17.15	10.43
	100	10	84 823	1 694 029	41.84	19.97
	150	10	85 027	3 228 899	68.77	25.48
T20I6D100K	50	20	99 913	916 315	27.84	18.34
	100	20	99 919	4 198 947	62.52	42.02
	150	20	99 924	7 479 490	stopped at 95 000 trans.	52.84
T20I18D100K	50	20	81 428	632 518	19.99	15.53
	100	20	81 472	2 485 199	46.10	30.50
	150	20	81 639	4 330 529	72.02	35.60
T40I10D100K	50	40	100 000	617 024	27.85	12.34
	100	40	100 000	5 561 767	72.28	55.61
	150	40	100 000	9 070 580	stopped at 90 000 trans.	67.64
T40I35D100K	50	40	85 445	470 245	20.75	11.00
	100	40	85 410	3 175 313	50.72	37.17
	150	40	85 646	5 574 661	88.04	43.68

Table 2. Experimental results on real and artificial datasets. For artificial ones, T denotes the average items per transaction, I the average length of maximal patterns and D the number of transactions. When I is close to T, *i.e.* when the dataset is dense, BoolLoader is more efficient than when I is smaller than T. All the given times include both system and user time.

BoolLoader on some known benchmarks [12]: *Mushroom, Chess, Connect, Pumsb*. With our current implementation, we can handle databases of about $D \times N = 10^7$ where D is the number of transactions and N the number of items.

Some datasets, such as Pumsb, seem to be intractable by BoolLoader. We have studied the evolution of the number of nodes according to the number of already processed transactions (Figure 5) and we have noticed a quite linear relationship between these two dimensions, except for mushroom. Concerning mushroom, we observe in Figure 5 that this dataset shows a particular evolution during the loading process. In fact, in mushroom, maximal itemsets are long (about 22 items [7]) which is quite the average length of a transaction. This means that in the corresponding BDD, many paths are shared and this explains the few number of nodes and the very low value of the sparseness coeffi cient of that dataset.

On the other hand, mushroom, connect and chess are known to be dense (with long maximal itemsets) and their spareseness coeffi cient is less than 10% whereas sparse synthetic datasets, such as T10I4D10KN1000, have high coeffi cient. In order to study the link between sparseness and our coeffi cient, we have generated several artifi cial datasets. Our study (Figure 5) shows that artifi cial databases do not behave as real databases (they are too smooth); this point has already been pointed out in [22].

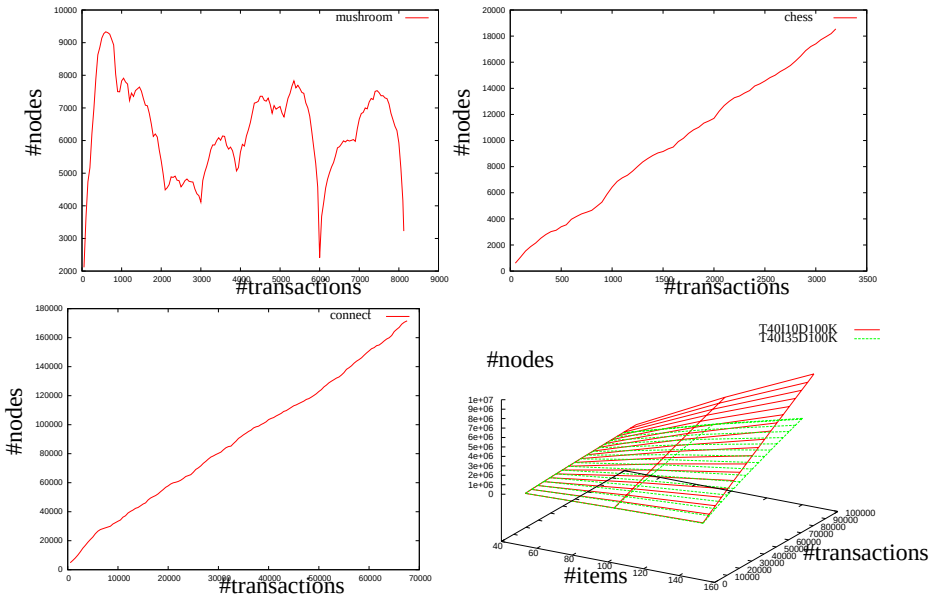


Fig. 5. Evolution of the number of nodes according to the number of transactions in the datasets: mushroom, connect, chess, T40I10D100K and T40I35D100K.

7 Conclusion and Future Works

In this paper we present BoolLoader, a tool for representing a transactional database by a BDD. Our aim when designing that tool was to investigate the use of such a data structure for data mining. For the time being, we have studied the feasability of that representation: it seems to be well suited for some databases but we have also given limits of that approach in terms of the number of items and the number of transactions. In our experiments no preprocessing has been done on the datasets. It could be interesting to find a "good ordering" of the variables to build a more condensed BDD, but it is known to be a NP-complete problem [11, 19] and heuristics have to be found. Beyond this study, we believe that BoolLoader and the Sparseness Coeffi cient that we have introduced could be an interesting estimator of the density of the database. Moreover, we have shown in [14] how to mine maximal frequent itemsets in datasets represented by a BDD. In the future, we would like to study other strategies for building a BDD from a transactional database but also to design clever algorithms to mine such data structure.

Acknowledgments The authors would like to thank the referees for their comments and suggestions.

References

1. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 1994.

2. S.B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27:509–516, 1978.
3. R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 85–93, Seattle, Washington, June 1998.
4. R.E. Bryant. Graph-based algorithms for boolean functions manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, August 1986.
5. D. Burdick, M. Calimlim, and 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.
6. E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi terminal binary decision diagrams: An efficient data structure for matrix representation. In *Int. Workshop on Logic Synth.*, pages P6a:1–15, 1993.
7. K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *Proceedings of 1st IEEE International Conference on Data Mining*, San Jose, November 2001.
8. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *Proc. of Int. Conf. on Management of Data*, pages 1–12, May 2000.
9. C.Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell Systems Technical Journal*, 38:985–999, July 1959.
10. 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.
11. S. Minato. *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Academic Publishers, 1996.
12. P. M. Murphy and D. W. Aha. *UCI Repository of Machine Learning Databases*. Machine-readable collection, Dept of Information and Computer Science, University of California, Irvine, 1995. [Available by anonymous ftp from `ics.uci.edu` in directory `pub/machine-learning-databases`].
13. R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and 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.
14. A. Salleb, Z. Maazouzi, and C. Vrain. Mining maximal frequent itemsets by a boolean approach. In *Proceedings of the 15th European Conference on Artifi cial Intelligence ECAI*, pages 385–389, Lyon, France, 2002.
15. A. Salleb and C. Vrain. Can We Load Efficiently Dense Datasets ? Technical Report RR-2004-02, LIFO, Université d'Orléans, 2004.
16. A. Savasere, E. Omiecinsky, and S. Navathe. An efficient algorithm for mining association rules in large databases. *In 21st Int'l Conf. on Very Large Databases (VLDB)*, 1995.
17. P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, , and D. Shah. Turbo-charging vertical mining of large databases. In *Proc. of Int.. Conf. on Management of Data*, 2000.
18. M. H. Stone. Boolean algebras and their relation to topology. *Proceedings of National Academy of Sciences*, 20:37–111, 1934.
19. S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *ISAAC: 4th International Symposium on Algorithms and Computation Algorithms)*, 1993.
20. M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *9th International Conference on Knowledge Discovery and Data Mining*, Washington, DC, August 2003.
21. M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, pages 283–296. AAAI Press, 1997.
22. Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 401–406, 2001.