A Method of Variable Ordering for Zero-suppressed Binary Decision Diagrams in Data Mining Applications

Haruya Iwasaki, Shin-ichi Minato and Thomas Zeugmann
Division of Computer Science
Hokkaido University
N-14, W-9, Sapporo 060-0814
Japan
{iwsk,minato,thomas}@ist.hokudai.ac.jp

Abstract

Recently, an efficient method of database analysis using Zero-suppressed Binary Decision Diagrams (ZBDDs) has been proposed. BDDs are a graph-based representation of Boolean functions, now widely used in system design and verification. Here we focus on ZBDDs, a special type of BDDs, which are suitable for handling large-scale combinatorial itemsets in transaction databases. The ZBDD size greatly depends on the variable ordering used. In this paper, we propose a new method of ZBDD variable ordering for itemset mining of large-scale transaction databases, and show experimental results.

1. Introduction

Discovering useful knowledge from large-scale databases has attracted a considerable attention during the last decade (cf., e.g., [11]). Frequent pattern mining is one of the fundamental data mining problems. Since the pioneering paper by Agrawal *et al.* [1] various algorithms have been proposed to solve the frequent pattern mining problem (cf., e.g., [12, 4]).

Recently, we have attacked the problem of efficiently generating the frequent patterns in a transaction database by using a data structure called Zero-suppressed Binary Decision Diagrams (abbr. ZBDDs), see [6][7, 8]. ZBDDs are a special case of Binary Decision Diagrams (abbr. BDDs) [2].

Using ZBDDs one can implicitly enumerate sets of combinations. Moreover, one can then perform efficiently various operations including the discovery and analysis of frequent patterns. It is known that the efficiency of these methods is greatly influenced by the variable ordering used in constructing the ZBDDs. So, the situation is comparable to BDDs, where the variable ordering also heavily influences the resulting size. As far as BDDs are concerned, the problem of finding a suitable variable ordering has been studied intensively (see, [3, 5, 9]) but for ZBDDs when used for mining transaction databases this line of research is still in its infancy. In the present paper, we propose a method for computing a variable ordering for ZBDDs which is based on the structure of database to be mined and present experimental results.

2. Database Representations Using ZBDDs

We consider databases of the following type. Let $M \neq \emptyset$ be any set. We refer to the elements of M as to *items*. In our examples below, we use $M = \{a, b, c\}$. Then the set of all possible combinations is the power set $\wp(M)$ of M. Any subset $\mathcal{C} \subseteq \wp(M)$ is said to be a set of combinations. The elements of a set of combinations are sets of items, e.g., $\{a, c\}$. To simplify notation, we write ac instead of $\{a, c\}$ and we refer to the elements of a set of combinations as to *tuples*. A transaction database is just a list of tuples.

In order to mine useful information from a transaction database it is very helpful to consider two types of *histograms*. The first type is called *tuple histogram*. A tuple histogram is a table having two columns. The first column contains all tuples that appear in the database and the second column contains the frequency of the tuple, i.e., a number describing how often the corresponding tuple appears in the database. The table in Fig. 3 is an example of a tuple histogram.

The second type of histogram is called *pattern histogram*. It contains a tuple and all subsets of it (again in the first column) and in the second column the number of appearances of the corresponding item sets in the database. For example, a and c each appear 8 times and b appears 10 times in the database used for constructing the table in Fig. 3.

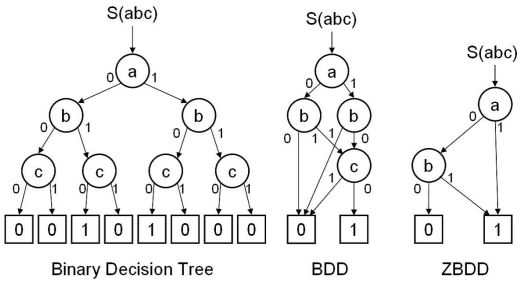


Figure 1. Binary Decision Tree, BDDs and ZBDDs

Histograms are important, since from a tuple histogram one can easily find frequent tuples and from a pattern histogram frequent patterns. Also, we mainly use the tuple histogram to compute from it the pattern histogram. Details are provided below. A typical data mining task is then to find all frequent patterns that are equal to or above some given threshold. For instance, suppose we are given the database from our example above and the threshold 10, then the data mining algorithm should return b. Given threshold 8, it should return a, b and c. Clearly, since the databases are usually very large, the main problem is how to do this efficiently.

Using the VSOP (Valued-Sum-Of-Products calculator) [7], we resolve this task as follows. Receiving a database and threshold as input, VSOP internally generates a ZBDD to represent the tuple histogram. Then, using the ZBDD-growth algorithm [8] developed by our group, a ZBDD is generated that represents all frequent patterns above or equal to the given threshold. In both steps the same variable ordering is used and this ordering heavily affects the size of the resulting ZBDDs (see Subsection 4.2). We continue with a more detailed description.

2.1. BDDs and ZBDDs

A BDD [2] is a graph representation for a Boolean function. An Example is shown in Fig. 1 for $S(a,b,c) = a\overline{b}\overline{c} \vee \overline{a}b\overline{c}$.

Given a variable ordering (in our example a, b, c), one can use Bryant's algorithm[2] to construct the BDD for any given Boolean function. For many Boolean functions appearing in practice this algorithm is quite efficient and the resulting BDDs are much more efficient representations than binary decision trees.

BDDs were originally invented to represent Boolean functions. But we can also map a set of combinations into Boolean space of n variables, where n is the cardinality of the item set M (see Fig. 2). So, one could also use BDDs to represent sets of combinations. However, one can even obtain a more efficient representation by using ZBDDs [6].

a	b	c	F	
0	0	0	0	
0	0	1	1	$\rightarrow c$
0	1	0	0	
0	1	1	1	$\rightarrow bc$
1	0	0	0	
1	0	1	0	
1	1	0	1	$\rightarrow ab$
1	1	1	1	$\rightarrow abc$

As a Boolean function:
$F = ab \vee \overline{a}c$

As a set of combinations:
$F = \{ab,\ abc,\ bc,\ c\}$

Figure 2. Correspondence of Boolean functions and sets of combinations.

If there are many similar combinations then many subgraphs are shared resulting in a smaller representation. ZBDDs have a special type of node deletion rule. All nodes whose 1-edge directly points to the 0-terminal node are deleted. Because of this, the nodes of items that do not appear in any sets of combinations are automatically deleted as shown in Fig.1.

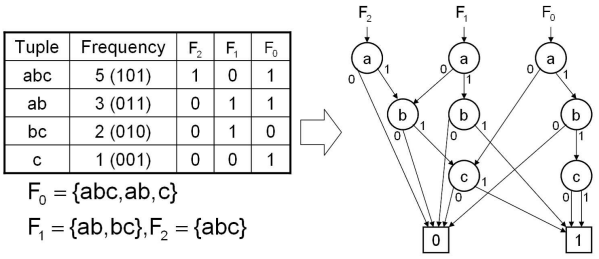


Figure 3. Representation of a tuple histogram

It remains to explain how to represent a tuple histogram by using ZBDDs. Since ZBDDs are representations of sets of combinations, a simple ZBDD distinguishes only the existence of each combination in the database. In order to represent the numbers of combination's appearances, we decompose the number into m-digits of ZBDD vector $\{F_0, F_1, \ldots, F_{m-1}\}$ to represent integers up to $(2^m - 1)$, as shown in Fig. 3. Namely, we encode the appearance numbers into binary digital code, where F_0 represents a set of tuples appearing odd times (LSB = 1), F_1 represents a set of combinations whose appearance number has a 1 in the second lowest bit, and similarly we define the set of each digit up to F_{m-1}. In the example of Fig. 3, the item set frequencies are decomposed as: $F_0 = \{abc, ab, c\}$, $F_1 = \{ab, bc\}$, $F_2 = \{abc\}$, and then each digit can be represented by a simple ZBDD. The three ZBDDs share their sub-graphs to one another.

By using the VSOP calculator, we can efficiently generate the tuple histograms on the transaction databases. However, if we have a tuple containing k items, then it contains 2^k many patterns. Thus, pattern histograms are exponentially larger than tuple histograms. If we use ZBDDs, we can represent similar patterns compactly by sharing nodes in the graph of the ZBDD. Thus, we are able to generate pattern histograms to some degree of scale. So the idea is to generate only the pattern histograms for the patterns appearing frequently, i.e., above or equal to the threshold given. This is done by using the ZBDD-growth algorithm (see Minato [8]).

3. ZBDD Variable Ordering for Database Representation

As described above, it is possible for us to represent histograms of frequent pattern sets compactly by using the ZBDD data structure. However, in the case of large-scale databases, it is difficult to generate the ZBDDs. For attacking this problem, we consider "the variable ordering of items." As already noted, the resulting ZBDD size is quite sensitive with respect to the underlying ordering of the variables. So, we aim to find a "good" variable ordering such that the resulting ZBDD size is close to the smallest size possible.

Clearly, if the cardinality of the item set M is small, then any ordering can be used. But if there are many items, then the exponential size of pattern histograms requires a very compact representation in order to be manageable. Thus, in such cases we can expect a huge benefit if we are able to find very compact ZBDD representations.

By default, the VSOP calculator uses the "late appearance upper ordering." That is, the item observed first in the database stays on the bottom of the ordering. The next item observed goes on top of it and so on. For example, if the database is given by the list ac, $befg$, ba, h, then the resulting ordering is h, g, f, e, b, c, a.

In the experiments we have performed previously for mining transaction databases by using our ZBDD data structure, we almost always used this ordering. The cost for computing this ordering are zero and quite often a relatively good ordering is obtained. However, this ordering solely depends on the ordering of records in the database, and thus may result also in a bad ordering. So, we looked for a different method which should result in a good ordering and which should be independent on the order of items in the database. The method obtained is, at least in spirit, similar to a method used for BDDs. Thus, we shortly describe this method for BDDs.

3.1. Dynamic Weight Assignment Method for Ordinary BDDs

In the field of logic VLSI circuit design, many researchers have dealt with the problem of finding good variable orderings for BDDs. In particular, Minato [9] proposed the so-called "dynamic weight assignment method" which turned out to be quite successful. Nevertheless, the problem of finding the optimal variable ordering for BDDs is known to be NP-complete [10].

For ordinary BDDs, two features of the variable ordering are known that affect the size of the resulting BDDs [3].

(1) Pairs of inputs having the local computability property should be kept close to one another in the ordering. This will then results in may shared subgraphs.

(2) Inputs having a strong influence to the output should be located at higher order.

Although one should try to come up with orderings obeying these two features, it may be difficult to do so, since these requirements are sometimes contradictory.

On the other hand, there are good heuristic methods computing the variable ordering by using information about the connectivity in circuits. Minato's [9] "dynamic weight assignment method" (abbr. DWA method) belongs to this group.

In the DWA method, in order to fulfill Feature (2) described above, one searches for inputs having strong power to control the output. This is done by analyzing the following.

- There are many fan-out's (there are many path to output).
- There are less steps until the output.
- There are less fan-in's on the path to the output.

For analyzing theses features, the following weight assignment method is used.

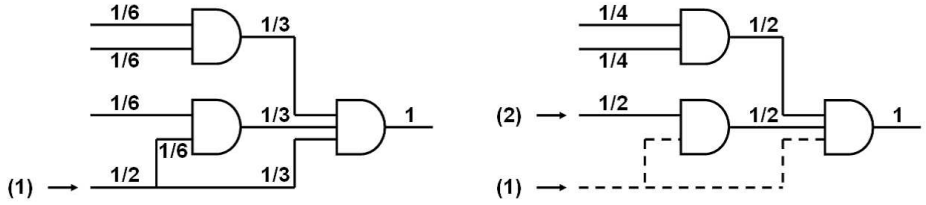
(1) Assign weight "1" to the output line.

(2) Carry the weights from output to input. If there are the gates having more than 1 inputs, then the weights are shared equally to inputs.

(3) For wires that are just connected, we add their weights.

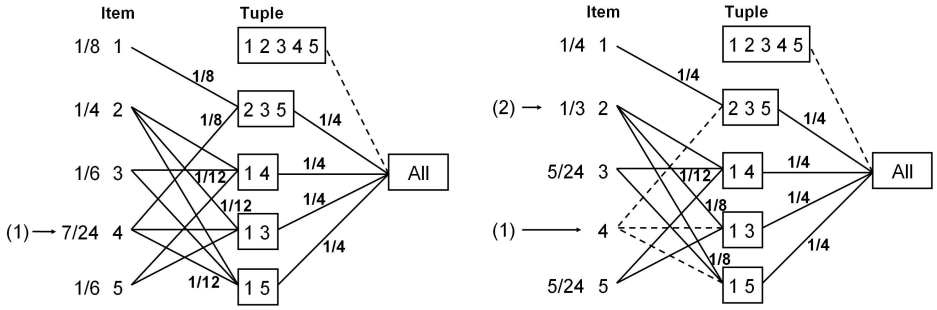
Fig. 4 exemplifies the method described above. Part (a) shows the initialization. The output gets weight 1. Since the output gate has fan-in 3, all its input wires get weight $1/3$.

We identify the input having maximal weight as the input having the strongest power to control output, and the



(a) First (b) Second

Figure 4. the Dynamic Weight Assignment Method for Logic Circuit



(a) First (b) Second

Figure 5. the DWA Method for Transaction Database

corresponding input variable goes to the top of the variable ordering.

Then we delete all the wires from this input to its sons and repeat the whole procedure (see Part (b) in Fig. 4). This repetition reflects the local computability, described above as Feature (1). Let n be the number of input variables and m the number of gates. Then the DWA method needs time $O(nm)$.

Next, we devise the DWA method to our problem of finding a good variable ordering for ZBDDs.

3.2. Applying the DWA Method to ZBDDs

The adaptation is done as follows (see Fig. 5 for an example). First, we construct a graph by introducing a node for each item. In Fig. 5 we have 5 items and the nodes are drawn on the left-hand side. Then, we draw a node for each tuple in the database and finally a node marked All. All is connected to each tuple. From each tuple we draw an edge to the each item *not* present in the tuple. (This is an important point, we consider that the items *not* present in the tuple have strong influences to the frequent patterns.) If there is a node in the tuple column not having an input, then it is deleted (in Fig. 5, Part (a), this is indicated by the dotted line form All to 12345). All gets weight 1, and the weight propagation is then done *mutatis mutandis* as in the original DWA algorithm. So, in our example, each solid edge from All to a tuple gets weight $1/4$. Node 235 has two inputs, thus each edge to an input now receives weight $1/8$ and so on. Item 4 has the highest weight, thus 4 goes to the top of the variable ordering.

Now, we continue as above. All edges to a son of node 4 are deleted and the process is repeated (see Part (b) in Fig. 5). Iterating this method results in a variable ordering only depending on the structure of the database.

4. Experimental Results

In our experiments we used Pentium4 CPU, SUSE Linux 9.3 as OS, and 512Mbyte main memory. The maximal value of ZBDD nodes is 10 millions. Our first experiment uses a mathematical example, while the remaining ones are performed by using practical benchmark examples. For showing the influence of variable orderings, we used four orderings, i.e., the ordering obtained by the DWA method for ZBDDs, the "More Frequency Upper Ordering," the "Less Frequency Upper Ordering," and the "Late Appearance Upper Ordering" already described. So, we have to describe the remaining two orderings. The "More Frequency Upper Ordering" is obtained by ordering the items with respect to their frequency of appearance. In this method more frequent items are located at higher order. "Less Frequency Upper Ordering" is just the opposite.

4.1. Results for Mathematical Database Examples

First, we consider a mathematical benchmark example. Given are $2n$ items $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$. The database is constructed by putting in row i the tuple consisting of all items except a_i and b_i (see below).

a_2	a_3	a_4	$\cdots$	a_n	b_2	b_3	b_4	$\cdots$	b_n
a_1	a_3	a_4	$\cdots$	a_n	b_1	b_3	b_4	$\cdots$	b_n
a_1	a_2	a_4	$\cdots$	a_n	b_1	b_2	b_4	$\cdots$	b_n
	$\vdots$				$\vdots$				$\vdots$
a_1	a_2	a_3	$\cdots$	a_{n-1}	b_1	b_2	b_3	$\cdots$	b_{n-1}

The variable orderings obtained for the DWA method and "Less Frequency Upper Ordering" are as follows.

the DWA method: $a_2\ b_2\ a_3\ b_3\ a_4\ b_4\ \cdots\ a_n\ b_n\ a_1\ b_1$

Less Frequency Upper Ordering:
$a_2\ a_3\ a_4\ \cdots\ a_n\ b_2\ b_3\ b_4\ \cdots\ b_n\ a_1\ b_1$

Note that the "More Frequency Upper Ordering" and "Late Appearance Upper Ordering" gave almost the same variable ordering than "Less Frequency Upper Ordering." The results obtained are displayed in Table 1 in dependence on the ordering and the number n. As we see, the variable ordering greatly affects the size of the resulting ZBDDs. Using the ordering obtained by the DWA method, the size of the ZBDDs grows proportionally to the number of items. On the hand, the size of the ZBDDs grows exponentially if the "Less Frequency Upper Ordering" is used. For an intuitive explanation, note that the DWA method returns an ordering such that the weight of the items is shared to closely related items. This shows that the local computability property is very essential for this database. Because the all items included in this database have the same frequency, also the other two ordering methods based on frequency give exponentially growing ZBDD size.

4.2. Results for Practical Benchmark Examples

Next, we provide experimental results for the effect of variable ordering using the DWA method on practical benchmark examples. For these experiments, we wrote a VSOP script for constructing the tuple histograms using the DWA method, and we picked up the frequent patterns from this script using the ZBDD-growth method. Furthermore, we counted the number of nodes of the generated ZBDDs. The results are displayed in Table 2. In this table, the number in "()" next to the name of the database is the threshold to represent the minimum frequency of patterns. Therefore, only the patterns having at least this threshold frequency are picked up.

Table 1. The Computation Time of VSOP Calculator for Mathematical Benchmark Examples

	the DWA Method		Less Freq. Upper Order.	
n	size	time(s)	size	time(s)
2	5	<0.1	5	<0.1
3	10	<0.1	12	<0.1
4	15	<0.1	25	<0.1
5	20	<0.1	50	<0.1
6	25	<0.1	99	<0.1
7	30	<0.1	196	<0.1
8	35	<0.1	389	<0.1
9	40	<0.1	774	<0.1
10	45	<0.1	1,543	<0.1
12	55	<0.1	6,153	<0.1
14	65	<0.1	24,587	<0.1
16	75	<0.1	98,317	<0.1
18	85	<0.1	393,231	0.456
20	95	<0.1	1,572,881	1.954
22	105	<0.1	6,291,475	8.482
24	115	<0.1	Memory overflow	

As our results show, the reduction effect obtained by using the variable ordering obtained by the DWA method is similar to the one obtained by using the ordering obtained via the "Less Frequency Upper Ordering" method. For the BMS-WebView-1 data set, the DWA method gave a better result than all other methods. In particular, the reduction is impressive when compared to the "More Frequency Upper Ordering" method.

Table 3 shows the computation time needed by the DWA method.

4.3. Discussion

As our experiments show, the variable ordering greatly affects the resulting ZBDD size. For the practical benchmark examples, the DWA method provided a better variable ordering than the "More Frequency Upper Ordering" method and the "Late Appearance Upper Ordering" method. However, there is hardly any difference between the DWA method and the "Less Frequency Upper Ordering" method. While the local computability property greatly influenced the ZBDD size for the mathematical benchmark examples, it has almost no effect for the practical bench-

Table 2. the Effect of the DWA Method for Transaction Database

	the DWA Method		More Freq. Upper Order.		Less Freq. Upper Order.		Late Appear. Upper Order.	
	size	time(s)	size	time(s)	size	time(s)	size	time(s)
chess (2,000)	1,422	5.8s	3,856	2,036.6s	1,415	5.8s	2,778	64.8s
mushroom (1)	16,403	1.0s	448,734	1.9s	15,131	1.0s	40,557	1.1s
connect (60,000)	348	27.5s	1,659	5,402.3s	348	27.5s	374	62.8s
BMS-WebView-1 (30)	106,920	98.8s	389,181	778.2s	109,989	103.1s	152,431	153.4s

Table 3. the Computation Time of the DWA Method

chess	2.8sec
mushroom	17.9sec
connect	219.6sec
BMS-WebView-1	1,255.0sec

mark examples. This is caused by the fact that the weights for the less frequent items are higher when using the DWA method. Thus, the "Less Frequent Upper Ordering" method and the DWA method gave almost similar results for the practical benchmark examples.

When the frequency of items affects the size of ZBDDs more than the local computability method, one should use the method based on the frequency, since its computation time is much lower.

Future work should focus on reducing the computation time of the DWA method and on further experiments with different databases.

Acknowledgment. This research was partially supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (B) on "ZBDD-based Database Analysis" (Shin-ichi Minato, Leader) 17300041, 2006.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[2] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, C-35(8):677–691, 1986.

[3] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvement of Boolean comparison method based on binary decision diagrams. In *IEEE International Conference on Computer-Aided Design, 1988, ICCAD-88, Digest of Technical Papers*, pages 2–5. IEEE Computer Society, 1988.

[4] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[5] S. Malik, A. R. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *IEEE International Conference on Computer-Aided Design, 1988, ICCAD-88, Digest of Technical Papers*, pages 6–9, 1988.

[6] S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993*, pages 272–277. ACM Press, 1993.

[7] S. Minato and H. Arimura. Efficient combinatorial item set analysis based on zero-suppressed bdds. In *Proc. of IEEE/IEICE/IPSJ International Workshop on Challenges in Web Information Retrieval and Integration (WIRI-2005)*, pages 3–10, 2005.

[8] S. Minato and H. Arimura. Frequent pattern mining and knowledge indexing based on zero-suppressed bdds. In *Proc. 5th International Workshop on Knowledge Discovery in Inductive Databases (KDID-2006)*, pages 83–94, 2006.

[9] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference. Orlando, Florida, USA, June 24-28, 1990*, pages 52–57. IEEE Computer Society Press, 1990.

[10] S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Algorithms and Computation, 4th International Symposium, ISAAC '93 Hong Kong, December 15-17, 1993 Proceedings*, volume 762 of *Lecture Notes in Computer Science*, pages 389–398. Springer, 1993.

[11] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, second edition*. Morgan Kaufmann, San Francisco, CA, USA, 2005.

[12] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.