TCS-TR-A-**07-25**

TCS Technical Report

Itemset Mining Based on Cofactor Implication

by

Shin-ichi Minato

Division of Computer Science

Report Series A

May 2, 2007

Hokkaido University
Graduate School of
Information Science and Technology

Email: minato@ist.hokudai.ac.jp

Phone: +81-011-706-7682

Fax: +81-011-706-7682

Itemset Mining Based on Cofactor Implication

Shin-ichi Minato
Division of Computer Science
Hokkaido University
North 14, West 9
Sapporo 060-0814, Japan

May 2, 2007

(Abstract) In this paper, we propose a new method for discovering hidden information from large-scale transaction databases by considering a property of *cofactor implication*. Cofactor implication is an extension or generalization of *symmetric itemsets*, which has been presented recently. Here we discuss the meaning of cofactor implication for the data mining applications, and show an efficient algorithm of extracting all non-trivial item pairs with cofactor implication by using Zero-suppressed Binary Decision Diagrams (ZBDDs). Finally, we show an experimental result to see how many itemsets can be extracted by using cofactor implication, compared with symmetric itemset mining. Our result indicates that the use of cofactor implication has a possibility of discovering a new aspect of structural information hidden in the databases.

1 Introduction

Discovering useful knowledge from large-scale databases has attracted a considerable attention during the last decade. Frequent pattern mining is one of the fundamental problems for knowledge discovery. Since the pioneering paper by Agrawal *et al.*[1], various algorithms have been proposed to solve the frequent pattern mining problem (cf., e.g., [15, 6]).

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 [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.

After generating frequent item set data, we sometimes faced with another problem that the results of itemsets are too large and complicated to retrieve useful information. It is important for practical data mining to extract the key structures from the itemset data. For example, closed/maximal itemset mining[16, 14, 13] extracts a partial set of itemsets with a special property, and it reveals a kind of structural information in the database. In this context, recently our research group have developed several new method such as *disjoint itemset decomposition*[9], and *symmetric itemset mining*[10]. Both the two methods are based on ZBDD representation for transaction databases, and efficiently find hidden structural information using set operations supported by ZBDD manipulation.

In this paper, we propose a new aspect of itemset mining by considering a property of *cofactor implication*. The cofactor implication is an extension or generalization of *symmetric itemset*[10]. Here we discuss the meaning of the cofactor implication for the data mining applications, and show an efficient algorithm of extracting all non-trivial item pairs with the cofactor implication by using ZBDD operations. Finally, we show an experimental result to see how many itemsets can be extracted by using the cofactor implication, compared with symmetric itemset mining. Our result indicates that the use of the cofactor implication has a possibility of discovering a new aspect of structural information hidden in the databases.

2 Database Representation Using ZBDDs

In this section, we first describe the database representation to be discussed. Here 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.

A *Binary Decision Diagram (BDD)* is a graph representation for a Boolean function. An Example is shown in Fig. 1 for $F(a, b, c) = a\overline{b}c \vee \overline{a}\overline{b}\overline{c}$.

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 *Zero-suppressed* BDDs (ZBDDs)[7].

If there are many similar combinations then the subgraphs are shared resulting in a smaller representation. In addition, ZBDDs have a special type of node deletion rule. As shown in Fig. 3, 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. This ZBDD reduction rule is extremely effective if we handle a set of sparse combinations. If the average appearance ratio of each item is 1%, ZBDDs are possibly more compact than ordinary BDDs, up to 100 times.

ZBDD representation has another good property that each path from the root node to the 1-terminal node corresponds to each combination in the set. Namely, the number of such paths in the ZBDD equals to the number of combinations in the set. This beautiful property indicates that, even if there are no equivalent nodes to be shared, the ZBDD structure explicitly stores all items of each combination, as well as using an explicit linear linked list data structure. In other words, (the order of) ZBDD size never exceeds the explicit representation. If more nodes are shared, the ZBDD is more compact than linear list.

Using a ZBDD-based data structure, we can manipulate compressed representation of large-scale transaction databases on the main memory. Recently, our research group has developed an efficient algorithm "ZBDD-growth"[11] to generate ZBDDs compactly representing all frequent itemsets for given databases. Our method is not only enumerating/listing the frequent patterns but also efficiently analyzing the huge size of mining results by using ZBDD operations. For example, extracting all patterns including a certain items, or computing the intersection/union/difference set for given two sets of patterns. The computation time of those operations does not directly depend on the number of patterns but almost linear to the (compressed) ZBDD size. It is an important advantage of using ZBDDs.

3 Cofactor Implication Based on ZBDDs

In this section, we present the definition of the cofactor implication and discuss the properties.

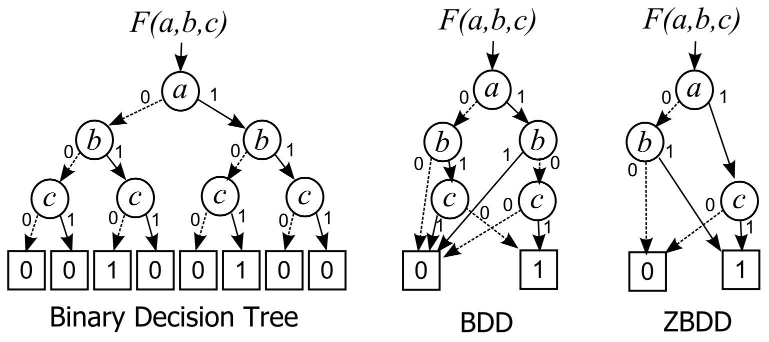

Figure 1: Binary Decision Tree, BDDs and ZBDDs

a	b	c	F	$\longrightarrow S$
0	0	0	0	
0	0	1	0	
0	1	0	1	$\longrightarrow b$
0	1	1	0	
1	0	0	0	
1	0	1	1	$\longrightarrow ac$
1	1	0	0	
1	1	1	0	

As a Boolean function:
$F = a\bar{b}c \vee \bar{a}b\bar{c}$

As a set of combinations:
$S = \{ac,\ b\}$

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

3.1 Symmetry of Items in Sets of Combinations

In our previous work[10], we defined symmetry of items in transaction databases, and we presented an efficient algorithm to find all symmetric items in a given database. Here we explain the meaning of symmetry of items. For example, we consider the following set of itemsets: $S = \{abc, acd, ad, bcd, bd, c, cd\}$. In this case, the item a and b are symmetric but the other pairs of variables are not symmetric. Figure 4 show a method of symmetry checking. First we classify the combinations into four categories: (1) S_{ab}, both a and b included, (2) $S_{a\bar{b}}$, only a is included, (3) $S_{\bar{a}b}$, only b is included, and (4) $S_{\overline{ab}}$, neither included. Namely, it can be written as:

$$S = a\ b\ S_{ab} \cup a\ S_{a\bar{b}} \cup b\ S_{\bar{a}b} \cup S_{\overline{ab}}$$

Then, we can determine the symmetry of a and b by comparing $S_{a\bar{b}}$ and $S_{\bar{a}b}$. If the two subsets are equivalent, a and b are exchangeable without any effect. For the above example, we can see that a and b are symmetric because $S_{a\bar{b}} = S_{\bar{a}b} = \{cd, d\}$.

By using quick equivalence checking based on ZBDD operations, we can efficiently detect symmetric itemsets in a large-scale transaction database. Previous experimental results[12] show that our method is applicable to a real-life biological database, which consists of 6,560 items of amino acid information related to influenza viruses. We succeeded in extracting all possible 64 groups of symmetric items from the database in a feasible computation time. Even if we do not know the actual meaning of each item, we can expect that the group of items would have somehow strong correlation if the symmetric property holds.

3.2 Symmetric Property and Cofactor Operation

Symmetric property is useful for extracting a kind of structural information from databases, however, in some cases, it is too strict to check $S_{a\bar{b}}$ and $S_{\bar{a}b}$ are completely identical. For example, if the database includes a bit of noise, the symmetric property is broken and we lose the chance to dis-

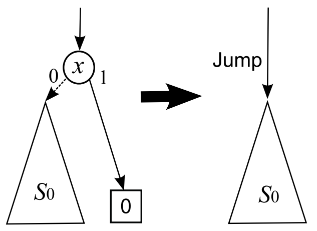

Figure 3: ZBDD reduction rule.

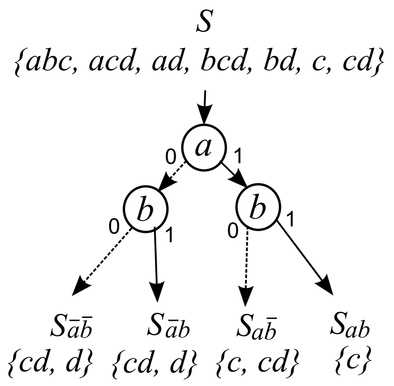

Figure 4: Symmetry checking based on ZBDD structure.

cover an interesting correlation. Therefore, we now consider to relax or to generalize the symmetric condition.

Let us consider the relationship between ZBDD data structure and symmetric property. A ZBDD decision node corresponds to the following expression of set operations.

$S = x\ S_x \cup S_{\overline{x}}$,

where x is the item symbol of the decision node, S_x and $S_{\overline{x}}$ are the subgraphs pointed by the 1-edge and the 0-edge, respectively. This operation, to obtain S_x and $S_{\overline{x}}$ from S, is called *cofactor* operation. It can be performed in a constant time (just looking pointers) if the item x is in the highest position in ZBDD variable ordering. Otherwise, the computation time is linear to the number of ZBDD nodes with the items higher than x.

In the transaction databases, S_x represents the set of item patterns co-occurring with x in some tuples, and $S_{\overline{x}}$ represents the set of patterns never co-occurring with x in the database. In this paper, we call S_x and $S_{\overline{x}}$ the *positive cofactor* and *negative cofactor*, respectively, if we distinguish the two kind of cofactors.

Symmetric items can be detected by generating the four sub-factors S_{xy}, $S_{x\overline{y}}$, $S_{\overline{x}y}$, $S_{\overline{xy}}$, which are the two stages of cofactor operations with respect to the items x and y. The condition $S_{x\overline{y}} \equiv S_{\overline{x}y}$ means that the sub-patterns co-occurring only with x and the sub-patterns co-occurring only with y are the completely identical.

It is a naive idea to relax the symmetric condition that $S_{x\overline{y}}$ and $S_{\overline{x}y}$ are not completely identical but $S_{x\overline{y}}$ implies $S_{\overline{x}y}$. We call this condition *cofactor implication*. If and only if the cofactor implication holds bidirectionally, symmetric property holds. Therefore, the cofactor implication is a partial condition for symmetry of items.

For example, in the following sets of combinations:

$S = \{abc, acd, ad, bc, bcd, bd, c, cd\}$,

the patterns co-occurring with a but not with b are $\{cd, d\}$, and the patterns co-occurring with b but not with a are $\{c, cd, d\}$. Thus, the cofactor implication holds from a to b.

Table 1: Conditions on cofactors and item correlations

Conditions	Correlations between the two items.
$S_{x\overline{y}} \equiv \emptyset$	x always occurs with y. (Direct implication.)
$S_{\overline{x}y} \equiv \emptyset$	y always occurs with x. (Direct implication.)
$S_{x\overline{y}} \equiv S_{\overline{x}y} \equiv \emptyset$	x, y co-occur with each other. (Complete co-occurrence.)
$S_{xy} \equiv \emptyset$	x, y are exclusive. (x and y never co-occur.)
$S_{\overline{xy}} \equiv \emptyset$	x, y are complementary. (At least either occurs.)
$S_{xy} \equiv S_{\overline{xy}} \equiv \emptyset$	x, y are exclusive and comlementary.
$S_{x\overline{y}} \subseteq S_{\overline{x}y}$	Cofactor implication holds from x to y.
$S_{\overline{x}y} \subseteq S_{x\overline{y}}$	Cofactor implication holds from y to x.
$S_{x\overline{y}} \equiv S_{\overline{x}y}$	x, y are symmetric.

3.3 Properties of Cofactor Implication

Let us discuss the meaning of the cofactor implication in transaction databases.

First, we can see that the cofactor implication is transitive as well as the symmetric condition. For example, if the cofactor implication holds from x to y and from y to z, then it also holds from x to z. Notice that the cofactor implication has the direction. This is different from the symmetric property.

Next, if the cofactor implication holds from x to y in a transaction database, all the patterns co-occurring with item x also co-occurs with item y. When the database represents a kind of knowledge, the cofactor implication represents a correlation between the two concepts used in the knowledge. Analyzing co-occurrence words is a basic method in the area of natural language processing[3]. Considering the cofactor implication would be an interesting idea to apply the recent state-of-the-art data mining techniques to the knowledge analysis and natural language processing.

Table 1 shows the list of various relationships of two items and their conditions based on the four cofactors, $S_{xy}, S_{x\overline{y}}, S_{\overline{x}y}, S_{\overline{xy}}$. In this table, the condition $S_{x\overline{y}} \equiv \emptyset$ (or $S_{\overline{x}y} \equiv \emptyset$) represents the direct implication property on the occurrences of x and y. This condition is commonly used to check the implication of items in transaction databases. It is contrastive that our cofactor implication is not directly checking the item occurrences but comparing the cofactors obtained by the items. In addition, $S_{x\overline{y}} \equiv \emptyset$ implies $S_{x\overline{y}} \subseteq S_{\overline{x}y}$, namely, the set of item pairs with the cofactor implication always includes ones with the ordinary direct implication. If we found a set of item pairs such that the cofactor implication holds but the ordinary implication does not hold, the result indicates a new aspect of item correlations which have never been considered.

In our definition of the cofactor implication, we should be careful that we do not use S_{xy} for the checking condition. There is the case that a sub-pattern P co-occurs with only x but not with only y, and that P still co-occurs with xy. If we consider such cases, the definition of cofactor implication can be modified as:

$S_{x\overline{y}} \subseteq (S_{\overline{x}y} \cup S_{xy})$.

However, if we use this modified definition, the property is not always transitive, and x and y are not always symmetric when the cofactor implication holds bidirectionally. In this paper, we consider only the simple definition: $S_{x\overline{y}} \subseteq S_{\overline{x}y}$, which has beautiful properties.

4 ZBDD-based algorithm for checking cofactor implication

In this section, we describe an efficient ZBDD-based algorithm of finding all the pairs of items such that the cofactor implication is holds for a given database S.

The “naive” checking method is to generate ZBDDs for $S_{x\overline{y}}$ and $S_{\overline{x}y}$ by cofactor operations for each pair of items x and y, and then compare the implication of the two cofactors. $S_{x\overline{y}}$ and $S_{\overline{x}y}$ can be computed in a constant time only if the two items

```
CoImplyAll($S$) {
  $I \leftarrow \emptyset$ ;
  for each item $x$ used in $S$ {
    $(S_{\overline{x}}, S_x) \leftarrow$ (Cofactors of $S$ by $x$) ;
    $I_x \leftarrow$ CoImplyItems($S_{\overline{x}}, S_x$) ;
    for each item $y \in I_x$
      $I \leftarrow I \cup \{$ "$x \rightarrow y$" $\}$ ;
  }
  return $I$ ;
}

CoImplyItems($S_{\overline{x}}, S_x$) {
  if ($S_x \equiv \emptyset$) return $\emptyset$ ;
  if ($S_x \equiv \{1\}$ && ($S_{\overline{x}} \equiv \emptyset$ | |$S_{\overline{x}} \equiv \{1\}$) ) return $\emptyset$ ;

  $I_x \leftarrow$ Cache($S_{\overline{x}}, S_x$) ;
  if ($I_x$ exists) return $I_x$ ;

  $y \leftarrow (S_{\overline{x}}, S_x).top$ ; /* Highest ordered item in $S_{\overline{x}}, S_x$ */
  $(S_{\overline{xy}}, S_{\overline{x}y}) \leftarrow$ (Cofactors of $S_{\overline{x}}$ by $y$) ;
  $(S_{x\overline{y}}, S_{xy}) \leftarrow$ (Cofactors of $S_x$ by $y$) ;

  if ($S_{x\overline{y}} \equiv \emptyset$)   $I_x \leftarrow$ CoImplyItems($S_{\overline{x}y}, S_{xy}$) ;
  else if ($S_{xy} \equiv \emptyset$)   $I_x \leftarrow$ CoImplyItems($S_{\overline{xy}}, S_{x\overline{y}}$) ;
  else  $I_x \leftarrow$ CoImplyItems($S_{\overline{x}y}, S_{xy}$) $\cap$ CoImplyItems($S_{\overline{xy}}, S_{x\overline{y}}$) ;

  if ($S_{x\overline{y}} \subseteq S_{\overline{x}y}$)   $I_x \leftarrow I_x \cup \{y\}$ ;

  Cache($S_{\overline{x}}, S_x$) $\leftarrow I_x$ ;
  return $I_x$ ;
}
```

Figure 5: Sketch of the cofactor implication checking algorithm.

x and y are at the nearly highest positions. Otherwise, the cofactor operation requires a linear time to the ZBDD size in average, so the total computation time becomes $O(n^2|S|)$, where n is the number of items and $|S|$ is the number of ZBDD nodes for S. To improve this time complexity, we developed an efficient recursive algorithm as presented in Fig. 5.

Our algorithm consists of the two parts. In the first part "CoImplyAll()," we choose one of item x used in S, and then call the other part "CoImplyItems()" to find the set of items I_x such that for any item $y \in I_x$ the cofactor implication from x to y holds.

The core of our algorithm, CoImplyItems(), is recursively implemented based on the cofactor operation with the highest ordered item y. The problem for $(S_{\overline{x}}, S_x)$ is divided into the two sub-problems for $(S_{\overline{xy}}, S_{x\overline{y}})$ and $(S_{\overline{x}y}, S_{xy})$, and the results of each sub-problems are computed recursively. The main result for $(S_{\overline{x}}, S_x)$ is obtained as the intersection of the results of the two sub-problems. Finally, the checking condition for item y itself, $S_{x\overline{y}} \subseteq S_{\overline{x}y}$ is determined and y is added into I_x if it holds. As well as the many of ZBDD-based operations, this algorithm uses a hash-based cache technique to avoid duplicated recursive calls. The number of recursive calls can be bounded by the number of ZBDD nodes if the hash table works well.

It is difficult to evaluate the exact complexity of CoImplyItems() due to the behavior of the hash-based cache. Since CoImplyItems() is called from CoImplyAll() for only n times, our algorithm can be up to n times faster than $O(n^2|S|)$ by the naive algorithm.

5 Experimental Results

We implemented our cofactor implication checking algorithm. The program is based on our own ZBDD package, and additional 70 lines of C++

Table 2: Database examples and their ZBDDs.

Name	#Items	#Records	ZBDD nodes	Time for gen.(sec)
mushroom	119	8,124	8,006	1.3
T10I4D100K	870	100,000	547,777	59.2
pumsb	2,113	49,046	1,749,775	166.7
BMS-Web-View-1	497	42,629	10,098	21.9
accidents	468	340,183	3,876468	127.5

Table 3: Experimental results for detecting item pairs.

Name	Symmetry check[10]		Cofactor implication check				
	Detected pairs	Time (sec)	All detected pairs	Time (sec)	Direct-imply	Others	Naive alg. Time(sec)
mushroom	19	0.6	1002	0.2	949	53	2.2
T10I4D100K	0	61.7	58	121.5	58	0	(>36,000)
pumsb	90	1152.0	39,323	642.1	39,322	1	(>36,000)
BMS-Web-View-1	6	30.2	206	4.3	181	25	423.0
accidents	11	18.0	4,509	303.0	4,509	0	(>36,000)

code for the symmetry checking algorithm. We used a 2.8GHz Pentium-4 PC, 1.5 GB of main memory, with SuSE Linux 9 and GNU C++ compiler. On this platform, we can manipulate up to 40,000,000 nodes of ZBDDs with up to 65,000 different items.

For evaluating the performance, we applied our method to the practical databases chosen from FIMI2003 benchmark set[5]. We first constructed a ZBDD for the set of all tuples in the database, and then apply our cofactor implication checking algorithm for the ZBDD. Table 2 shows the specifications of database examples and the sizes of ZBDDs for them. In our experiments, so far we do not consider the number of frequency of each tuple, only dealing with binary information.

The experimental results are shown in Table 3. In this table, we show our previous results of symmetry checking[10] and this paper's results of cofactor implication checking. The numbers of the cofactor implications are classified into the two categories: the ordinary direct implications and the others, which are newly detected ones. We can observe that much larger number of the cofactor implications are detected compared with the symmetric ones. This is a reasonable result since the cofactor implication is a partial condition of the symmetry.

On the computation time, our implementation of cofactor implication checking is competitive to the symmetric checking algorithm, and in some examples, our method is even faster than symmetric checking. Since our results also including all the symmetric item pairs, we can use the cofactor implication checking algorithm as an alternative method for the symmetric checking. We also compared the computation time with the naive algorithm to check all the n^2 pairs of items. Our algorithm is 10 to 100 times faster than the naive one.

In our experiments, we detected a small number of the cofactor implications which are not the ordinary direct implications. Those item pairs have no direct relationship of implication, but the implication property holds between the co-occurring subpatterns. Figure 6 and 7 illustrate the diagrams of those cofactor implications generated by graph drawing software *graphviz*[4]. In the diagrams, the two nodes mutually connected indicate the symmetric item groups. We can observe a number of correlated item clusters based on the connecting edges of the cofactor implications. Those clusters possibly represent interesting structural information hidden in the database, which has not been

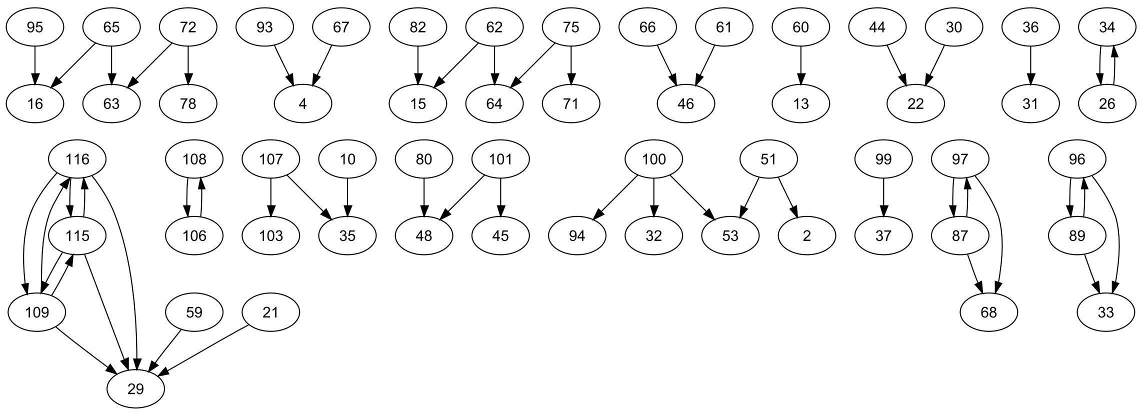

Figure 6: Cofactor implication diagram of “mushroom” (excluding direct implications).

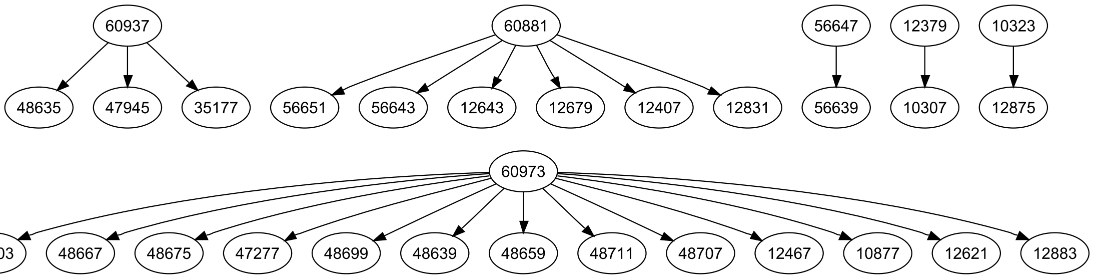

Figure 7: Cofactor implication diagram of “BMS-Web-View-1”(excluding direct implications).

discovered before.

6 Conclusion

In this paper, we proposed a new aspect of itemset mining based on the cofactor implication. We defined the cofactor implication as a relaxation of the symmetric condition, and discuss the meaning and the properties of the cofactor implication in the transaction databases. We also presented an efficient checking algorithm using ZBDD data structure, and showed the experimental results for the actual database benchmark examples. The experimental results show that our method is very powerful and will be useful for discovering hidden interesting information in the given data. As our future work, we will apply our method to real-life data mining problems, in order to compare our result of the item correlations with the original meaning of the items in the databases.

Analyzing co-occurrence words is a basic method in the area of natural language processing. Our result indicates a new possibility of applying the recent state-of-the-art data mining techniques to the knowledge analysis and natural language processing.

Acknowledgment

The author would like to thank Thomas Zeugmann, Hiroki Arimura, and Makoto Haraguchi of Hokkaido University for their valuable discussions and comments. 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,” 17300041, 2007, and Grant-in-Aid for Specially Promoted Research on “Semi-Structured Data Mining,” 17002008.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items

in large databases. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data, Vol. 22(2) of SIGMOD Record*, pages 207–216, 1993.

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

[3] I. Dagan, L. Lee, and F. Preira. Similarity-based models of word cooccurrence probabilities. *Machine Learning*, 34(1-3):43–69, 1999.

[4] E. Gansner, E. Koutsofios, S. North, and K. Vo. A technique for drawing directed graphs. *IEEE Trans. Software Eng.*, 19(3):214–230, 1993.

[5] B. Goethals and M. J. Zaki. Frequent itemset mining dataset repository, 2003. Frequent Itemset Mining Implementations (FIMI'03), `http://fimi.cs.helsinki.fi/data/`.

[6] 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.

[7] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of 30th ACM/IEEE Design Automation Conference*, pages 272–277, 1993.

[8] S. Minato. Zero-suppressed BDDs and their applications. *International Journal on Software Tools for Technology Transfer (STTT), Springer*, 3(2):156–170, 2001.

[9] S. Minato. Finding simple disjoint decompositions in frequent itemset data using zero-suppressed BDDs. In *Proc. IEEE ICDM 2005 workshop on Computational Intelligence in Data Mining*, pages 3–11, 11 2005. ISBN-0-9738918-5-8.

[10] S. Minato. Symmetric item set mining based on zero-suppressed BDDs. In *the 9th International Conference on Discovery Science (DS-2006), (LNAI 4265, Springer)*, pages 321–326, 10 2006.

[11] S. Minato and H. Arimura. Frequent pattern mining and knowledge indexing based on zero-suppressed BDDs. In *The 5th International Workshop on Knowledge Discovery in Inductive Databases (KDID'06)*, pages 83–94, 9 2006.

[12] S. Minato and K. Ito. Symmetric item set mining method using zero-suppressed BDDs and application to biological data. *Trans. of the Japanese Society of Artificial Intelligence*, 22(2):156–164, 2007.

[13] T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Proc. the 8th International Conference on Discovery Science 2004 (DS-2004)*, 2004.

[14] T. Uno, Y. Uchida, T. Asai, and H. Arimura. LCM: an efficient algorithm for enumerating frequent closed item sets. In *Proc. Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, 2003. `http://fimi.cs.helsinki.fi/src/`.

[15] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(2):372–390, 2000.

[16] M. J. Zaki and C. Hsiao. CHARM: an efficient algorithm for closed itemset mining. In *2nd SIAM International Conference on Data Mining (SDM'02)*, pages 457–473, 2002.