TCS-TR-A-06-22

TCS Technical Report

Symmetric Item Set Mining Method Using ZBDDs and Application to Biological Data

by

Shin-ichi Minato and Kimihito Ito

Division of Computer Science

Report Series A

November 11, 2006

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

Symmetric Item Set Mining Method Using ZBDDs and Application to Biological Data

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

Kimihito Ito
Research Center for Zoonosis Control
Hokkaido University
North 18, West 9
Sapporo 060-0818, Japan

November 11, 2006

(Abstract) In this paper, we present a method of finding symmetric items in a combinatorial item set database. The techniques for finding symmetric variables in Boolean functions have been studied for long time in the area of VLSI logic design, and the BDD (Binary Decision Diagram) -based methods are presented to solve such a problem. Recently, we have developed an efficient method for handling databases using ZBDDs (Zero-suppressed BDDs), a particular type of BDDs. In our ZBDD-based data structure, the symmetric item sets can be found efficiently as well as for Boolean functions. We implemented the program of symmetric item set mining, and applied it to actual biological data on the amino acid sequences of influenza viruses. We found a number of symmetric items from the database, some of which indicate interesting relationships in the amino acid mutation patterns. The result shows that our method is helpful for extracting hidden interesting information in real-life databases.

1 Introduction

Frequent item set mining is one of the fundamental techniques for knowledge discovery. Since the introduction by Agrawal et al.[1], the frequent item set mining and association rule analysis have been received much attentions from many researchers, and a number of papers have been published about the new algorithms or improvements for solving such mining problems[3].

After generating frequent item set data, we sometimes faced with the problem that the results of item sets are too large and complicated to retrieve useful information. Therefore, it is important for practical data mining to extract the key structures from the item set data. Closed/maximal item set mining[16] is one of the useful methods to find important item sets. Disjoint decomposition of item set data[13] is another powerful method for extracting hidden structures from frequent item sets.

In this paper, we propose one more interesting method for finding hidden structure from large-scale item set data. Our method is based on the symmetry of items. It means that the exchange of a pair of symmetric items has completely no effect for the database information. This is a very strict property and it will be a useful association rule for the database analysis.

The symmetry of variables is a fundamental concept in the theory of Boolean functions, and the method of symmetry checking has been studied for long time in VLSI logic design area. There are some state-of-the-art algorithms[14, 7] using BDDs (Binary Decision Diagrams)[2] to solve such a problem. The BDD-based techniques can be applied to data mining area. Recently, we found that ZBDDs (Zero-suppressed BDDs)[9] are very suitable for representing large-scale item set data used in transaction database analysis[12].

In this paper, we discuss the property of symmetric items in transaction database, and then present an efficient algorithm to find all symmetric item sets using ZBDDs. We also show the experimental result for an actual biological database on the amino acid sequences of influenza viruses[8][6]. We found a number of symmetric items from the database, some of which indicate an interesting relationship of amino acid mutation patterns. The result shows that our method is helpful for extracting hidden information in real-life databases.

2 Preliminaries

Here we briefly describe the basic techniques of BDDs and ZBDDs for representing combinatorial item sets efficiently.

2.1 BDDs

BDD (Binary Decision Diagram) is a directed graph representation of the Boolean function, as illustrated in Fig. 1(a). It is derived by reducing a binary tree graph representing recursive *Shannon's expansion*, indicated in Fig. 1(b). The following reduction rules yield a *Reduced Ordered BDD (ROBDD)*, which can efficiently represent the Boolean function. (see [2] for details.)

- Delete all redundant nodes whose two edges point to the same node. (Fig. 3(a))
- Share all equivalent sub-graphs. (Fig. 3(b))

ROBDDs provide canonical forms for Boolean functions when the variable order is fixed. Most researches on BDDs are based on the above reduction rules. In the following sections, ROBDDs will be referred to as BDDs (or ordinary BDDs) for the sake of simplification.

As shown in Fig. 2, a set of multiple BDDs can be shared each other under the same fixed variable ordering. In this way, we can handle a number of Boolean functions simultaneously in a monolithic memory space.

Using BDDs, we can uniquely and compactly represent many practical Boolean functions including AND, OR, parity, and arithmetic adder functions. Using Bryant's algorithm[2], we can efficiently construct a BDD for the result of a binary logic operation (i.e. AND, OR, XOR), for given a pair of operand BDDs. This algorithm is based on hash table techniques, and the computation time is almost linear to the data size unless the data overflows the main memory. (see [10] for details.)

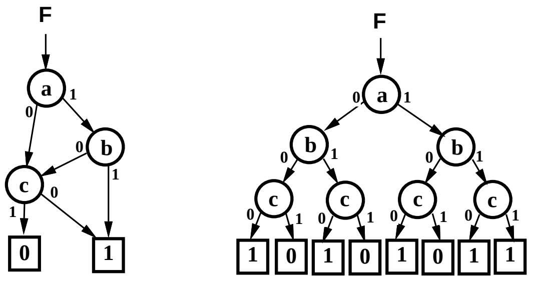
Based on these techniques, a number of BDD packages have been developed in 1990's and widely used for large-scale Boolean function manipulation, especially popular in VLSI CAD area.

2.2 ZBDDs and combinatorial item sets

BDDs are originally developed for handling Boolean function data, however, they can also be used for implicit representation of combinatorial item sets. Here we call "combinatorial item set" for a set of elements each of which is a combination out of n items. This data model often appears in real-life problems, such as combinations of switching devices(ON/OFF), fault combinations, and sets of paths in the networks.

A combination of n items can be represented by an n-bit binary vector, $(x_1 x_2 \dots x_n)$, where each bit, $x_k \in \{1, 0\}$, expresses whether or not the item is included in the combination. A set of combinations can be represented by a list of the combination vectors. In other words, a combinatorial item set is a subset of the power set of n items.

A combinatorial item set can be mapped into Boolean space by using n-input variables for each bit of the combination vector. If we choose any one combination vector, a Boolean function determines whether the combination is included in the combinatorial item set. Such Boolean functions are called *characteristic functions*. For example, the left side of Fig. 5 shows a truth-table representing a Boolean function $(ab\overline{c}) \vee (\overline{b}c)$, but also represents a combinatorial item set $\{ab, ac, c\}$. Using BDDs for characteristic functions, we can implicitly and compactly represent combinatorial item sets. The logic operations AND/OR for Boolean functions correspond to the set operations intersection/union for combinatorial item sets. By using BDDs for characteristic functions, we can manipulate combinatorial item sets efficiently. They can be generated and manipulated within a time roughly



(a) BDD. (b) Binary tree.

Figure 1: BDD and binary tree: $F = (a \wedge b) \vee \overline{c}$.

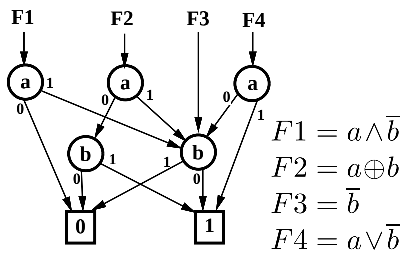
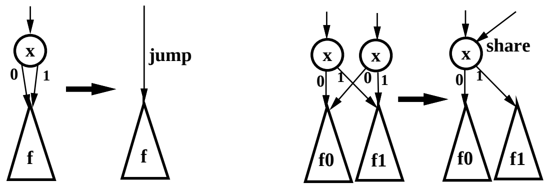


Figure 2: Shared multiple BDDs.



(a) Node deletion. (b) Node sharing.

Figure 3: Reduction rules of ordinary BDDs

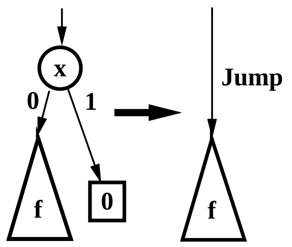


Figure 4: ZBDD reduction rule.

proportional to the BDD size. When we handle many combinations including similar patterns (sub-combinations), BDDs are greatly reduced by node sharing effect, and sometimes an exponential reduction benefit can be obtained.

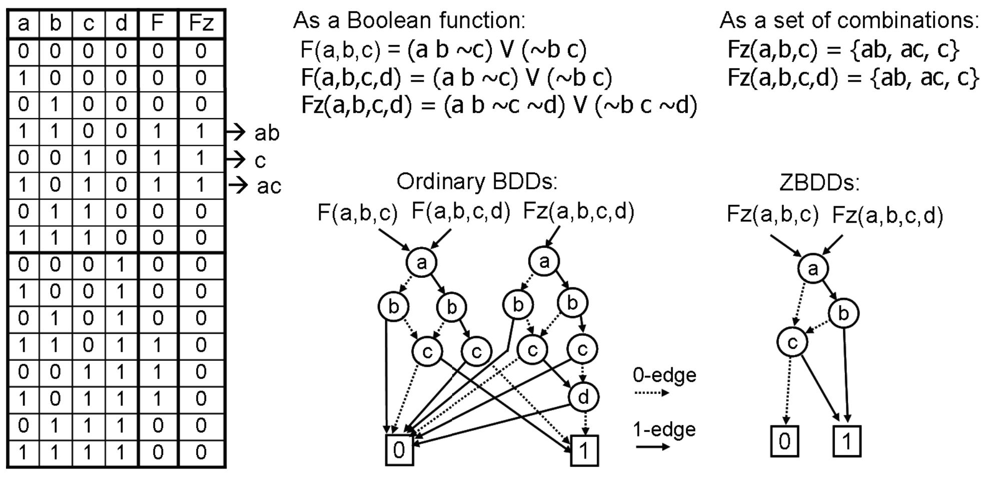
Zero-suppressed BDD (ZBDD)[9] is a special type of BDDs for efficient manipulation of combinatorial item sets. ZBDDs are based on the following special reduction rules.

- Delete all nodes whose 1-edge directly points to the 0-terminal node, and jump through to the 0-edge's destination, as shown in Fig. 4.
- Share equivalent nodes as well as ordinary BDDs.

Notice that we do not delete the nodes whose two edges point to the same node, which used to be deleted by the original rule. The zero-suppressed deletion rule is asymmetric for the two edges, as we do not delete the nodes whose 0-edge points to a terminal node. It is proved that ZBDDs also give canonical forms as well as ordinary BDDs under a fixed variable ordering.

Here we summarize the features of ZBDDs.

- In ZBDDs, the nodes of irrelevant items (never chosen in any combination) are automatically deleted by ZBDD reduction rule. In ordinary BDDs, irrelevant nodes still remain and they may spoil the reduction benefit of sharing nodes. An example is shown in Fig. 5. In this case, the item d is irrelevant, but ordinary BDD for characteristic function $Fz(a, b, c)$ and $Fz(a, b, c, d)$ become different forms. On the other hand, ZBDDs for $Fz(a, b, c)$ and $Fz(a, b, c, d)$ become identical forms and completely shared.
- 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. In ordinary BDDs, this property does not always hold.



a	b	c	d	F	Fz	
0	0	0	0	0	0	
1	0	0	0	0	0	
0	1	0	0	0	0	
1	1	0	0	1	1	→ ab
0	0	1	0	1	1	→ c
1	0	1	0	1	1	→ ac
0	1	1	0	0	0	
1	1	1	0	0	0	
0	0	0	1	0	0	
1	0	0	1	0	0	
0	1	0	1	0	0	
1	1	0	1	1	0	
0	0	1	1	1	0	
1	0	1	1	1	0	
0	1	1	1	0	0	
1	1	1	1	0	0	

As a Boolean function:
F(a,b,c) = (a b ~c) V (~b c)
F(a,b,c,d) = (a b ~c) V (~b c)
Fz(a,b,c,d) = (a b ~c ~d) V (~b c ~d)
As a set of combinations:
Fz(a,b,c) = {ab, ac, c}
Fz(a,b,c,d) = {ab, ac, c}
Ordinary BDDs:
F(a,b,c) F(a,b,c,d) Fz(a,b,c,d)
ZBDDs:
Fz(a,b,c) Fz(a,b,c,d)
0-edge
1-edge
a
b
c
d
0
1

Figure 5: Effect of ZBDD reduction rule.

- When no equivalent nodes exist in a ZBDD, that is the worst case, the ZBDD structure explicitly stores all items in all combinations, as well as using an explicit linear linked list data structure. Namely, (the order of) ZBDD size never exceeds the explicit representation. If more nodes are shared, the ZBDD is more compact than linear list.

The detailed techniques of ZBDD manipulation are described in the articles[9, 11]. A typical ZBDD package supports cofactoring operations to traverse 0-edge or 1-edge, and binary operations between two combinatorial item sets, such as union, intersection, and difference. The computation time for each operation is almost linear to the number of ZBDD nodes related to the operation.

3 Symmetric item sets in transaction databases

3.1 Symmetry of variables in Boolean functions

The symmetry is a fundamental concept in the theory of Boolean functions. A symmetric Boolean function means that any exchange of input variables has no effect for the output value. In other words, the output value is decided only by the total number of true assignments in the n-input variables. The parity check functions and threshold functions are typical examples of symmetric functions.

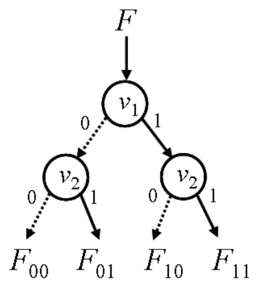


Figure 6: Four sub-functions for symmetry checking.

When the function is not completely symmetric, we sometimes find partial groups of symmetric variables. If two variables are exchangeable without any output change, we call them symmetric variables in the function. An obvious property holds that if the pairs (a, b) and (a, c) are both symmetric, then any pair in (a, b, c) is symmetric.

As finding symmetric variables leads to compact logic circuits, it has been studied for long time in VLSI logic design area. In order to check the symmetry of the two variables v_1 and v_2 in the function F, as shown in Fig. 6, we first extract four sub-functions: F_{00}, F_{01}, F_{10}, and F_{11} by assigning all combinations of constant values 0/1 into v_1 and v_2. We then compare of F_{01} and F_{10}. If the two are equivalent, we can see the two variables

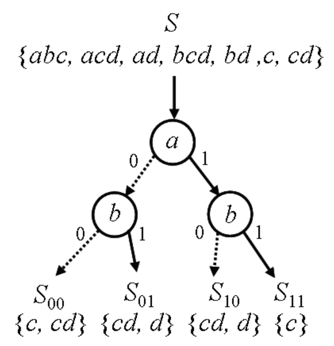


Figure 7: Four sub-combinations for symmetry checking.

are symmetric. In principle, we need $n(n-1)/2$ times of symmetry checks for all possible variable pairs. There have been proposed some state-of-the-art algorithms[14, 7] using BDDs (Binary Decision Diagrams)[2] to solve such a problem efficiently.

3.2 Symmetric Items in combinatorial item sets

Here we discuss the symmetry of items in a combinatorial item set. For example we consider the following combinatorial item set:

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

In this case, the item a and b are symmetric but the other pairs of variables are not symmetric. The symmetry can be confirmed as shown in Fig. 7. First we classify the combinations into four categories: (1) both a and b included, (2) only a is included, (3) only b is included, and (4) neither included. Namely, it can be written as: $S = abS_{11} \cup aS_{10} \cup bS_{01} \cup S_{00}$. Then, we can determine the symmetry of a and b by comparing S_{10} and S_{01}. If the two subsets are equivalent, a and b are exchangeable. For the above example, $S_{11} = \{c\}, S_{10} = \{cd, d\}, S_{01} = \{cd, d\}$, and $S_{00} = \{c, cd\}$. We can see a and b are symmetric as $S_{10} = S_{01}$.

Even if we do not know the actual meaning of the item a and b in the original database, we can expect that a and b would have somehow strong relationship if the symmetric property holds. It is a kind of hidden information. It would be a useful and interesting task to find all possible symmetric item sets from the given databases. This method can be used not only for original database but also for frequent item set data to find some relationships between the items.

4 ZBDD-based algorithm for finding symmetric item sets

As shown in article[12], the ZBDD-based data structure is quite effective (exponentially in extreme cases) for handling transaction databases, especially when the item sets include many similar partial combinations. Now we show an efficient algorithm of finding symmetric item sets based on ZBDD operations.

First we explain the *cofactor* operation on ZBDDs. Cofactor(S, v) classifies a combinatorial item set S into the two subsets, one of which includes the item v and the other does not. Namely, it extracts S_1 and S_0 such that $S = vS_1 \cup S_0$. If the item v is the top (highest ordered) item in the ZBDD, then S_1 and S_0 are the two sub-graphs pointed by 1-edge and 0-edge of the top decision node, and the cofactor operation can be done in a constant time. Therefore, if the item v_1 and v_2 are the first and second top items in the ZBDD, the symmetry checking is quite easy because S_{10} (subset with v_1 but not v_2) and S_{01} (subset with v_2 but not v_1) can be extracted and compared in a constant time.

This “naive” checking method works quite efficiently only if the v_1 and v_2 are at the highest positions. Otherwise, we have to generate temporary ZBDDs for S_{10} and S_{01} by cofactor operations. If S_{10} and S_{01} are quite different, we may easily find the asymmetry of two items by checking a small part of ZBDDs. However, the naive method always generates the whole ZBDDs for S_{10} and S_{01} and then compare them. To address this inefficiency, we developed an efficient recursive algorithm as presented in 8.

In this algorithm, first we get the top item t in the ZBDD S, and extract S_1 and S_0 as the cofactors of S by t. We then recursively check the symmetry of (v_1, v_2) for each subset S_1 and S_0, and if they are symmetric for the both, we can see they are symmetric for S.

```
SymChk($S, v_1, v_2$) /* Assume $v_1$ higher than $v_2$ in the ZBDD ordering.*/
{
  if ($S = 0$ or $S = 1$) return 1 ;
  $r \leftarrow$ Cache($S, v_1, v_2$) ;
  if ($r$ exists) return $r$ ;
  $t_1 \leftarrow S.top$ ; /* Top item in $S$ */
  if ($t_1$ higher than $v_1$)
    $(S_1, S_0) \leftarrow$ (Cofactors of $S$ by $t_1$) ;
    $r \leftarrow$ SymChk($S_1, v_1, v_2$) && SymChk($S_0, v_1, v_2$) ;
  else
    $(S_1, S_0) \leftarrow$ (Cofactors of $S$ by $v_1$) ;
    $t_2 \leftarrow$ Max($S_1.top, S_0.top$) ; /* Top item in $S_1, S_0$ */
    if ($t_2$ higher than $v_2$)
      $(S_{11}, S_{10}) \leftarrow$ (Cofactors of $S_1$ by $t_2$) ;
      $(S_{01}, S_{00}) \leftarrow$ (Cofactors of $S_0$ by $t_2$) ;
      $r \leftarrow$ SymChk($(t_2 S_{11} \cup S_{10}), t_2, v_2$) && SymChk($(t_2 S_{01} \cup S_{00}), t_2, v_2$) ;
    else
      $(S_{11}, S_{10}) \leftarrow$ (Cofactors of $S_1$ by $v_2$) ;
      $(S_{01}, S_{00}) \leftarrow$ (Cofactors of $S_0$ by $v_2$) ;
      $r \leftarrow (S_{10} = S_{01})$? 1 : 0 ;
    endif
  endif
  Cache($S, v_1, v_2$) $\leftarrow r$ ;
  return $r$ ;
}
```

Figure 8: Sketch of the symmetry checking algorithm.

This procedure may require an exponential number of recursive calls in terms of the number of items higher than v_1, v_2 in the ZBDD, however, we do not have to execute the procedure twice for the same ZBDD node because the results will be the same. Therefore, the number of recursive calls is bounded by the ZBDD size, by using a hash-based cache to save the result of procedure for each ZBDD node. In addition, if we found the two items are asymmetric either for S_1 or S_0, we may immediately quit the procedure and conclude they are asymmetric for S.

In our checking algorithm, the cofactor operation is always applied to the highest ordered items, and each recursive procedure can be executed in a constant time. Thus, the total computation time is bounded by $O(|G|)$, where $|G|$ is the ZBDD size for S. Repeating this procedure for all item pairs in S, we can extract all possible symmetric item sets in $O(n^2|G|)$ time, where n is number of items. The time will be shorter in practice because the most of item pairs are asymmetric in usual cases. In addition, the benefit of hash-based cache can be shared in the repetition of checking different item pairs.

Lastly we note that there have been several efficient symmetry checking algorithms for Boolean function data using ordinary BDDs[14, 7], but it is not trivial to apply the techniques to the ZBDDs since the primitive operations of ZBDDs are different from those of ordinary BDDs. Our work is the first proposal of the efficient symmetry checking algorithm for combinatorial item sets using ZBDDs.

5 Implementation and Application to biological data

We implemented our symmetric checking algorithm. The program is based on our own ZBDD package, and additional 70 lines of C++ code for the symmetry checking algorithm. We used a Pentium-4 PC, 800MHz, 1.5GB of main memory, with SuSE Linux 9. We can manipulate up to 20,000,000 nodes of ZBDDs in this PC.

Table 1: Experimental result for basic performance evaluation

Data name	#Item	#Record	#Tuple	ZBDD nodes	Time(sec) for ZBDD gen.	Sym. pairs	Time(sec) for sym.chk.
mushroom	119	8,124	8,124	8,006	1.1	19	0.6
T10I4D100K	870	100,000	89,135	547,777	59.2	0	61.7
pumsb	2,113	49,046	48,474	1,749,775	166.7	90	1,152.0
BMS-Web-View-1	497	59,602	18,473	42,629	24.9	6	30.2
accidents	468	340,183	339,898	3,876,468	127.5	11	18.0

Table 2: Comparison to naive checking method.

Data name	Time(sec) for sym.chk.	
	our method	naive method.
mushroom	0.6	1.2
T10I4D100K	61.7	28,482.0
pumsb	1,152.0	(> 24h)
BMS-Web-View-1	30.2	251.8
accidents	18.0	53,290.3

5.1 Experiment for basic performance evaluation

For evaluating the basic performance, we applied our method to the practical transaction databases chosen from FIMI2003 benchmark set[4]. We first constructed a ZBDD for the set of all tuples in the database, and then apply our symmetry checking algorithm for all item pairs. The results are shown in Table 1. "Sym. pairs" shows the number of symmetric pairs we found. Our result demonstrates that we succeeded in extracting all symmetric item sets for a practical size of databases within a feasible computation time. We can see that no symmetric pairs are found in "T10I4D100K." It is a reasonable result because this data is randomly generated and there is no strong relationship between any pair of items.

Table 2 shows the comparison of our symmetry checking algorithm to the "naive" method, which always generates the whole ZBDDs for S_{10} and S_{01} and then compare them. The differences of computation time are more than hundred times in larger instances. This result shows that our recursive checking algorithm is remarkably effective to find symmetric item pairs.

5.2 Experiment of biological data analysis

Hokkaido University Research Center for Zoonosis Control is conducting a research project for analyzing the mutations in the hemagglutinins of influenza viruses, in order to predict possible structural changes in the future influenza viruses[6]. The hemagglutinin (HA) is the major surface glycoprotein of influenza viruses and plays an important role in virus entry into host cells. The amino acid sequences of HAs mutate every year, and the continuously cause epidemic in the world.

We applied our method to a real-life biological data, the amino acid sequences of the HA of human influenza viruses.

The data we used are the 1,657 instances of amino acid sequences of type "H3N2" HAs of human influenza viruses, isolated during 1968 to 2006 in the world. Each sequence has 328 positions of amino acids, and each position has one of the 20 amino acid types: {R, K, H, P, A, L, G, V, I, W, M, S, Y, Q, T, F, N, C, E, D}. Namely, the total combinatorial space is as many as 20^{328}. The sequence data also has a field of the year when the virus was isolated. The data is available at the public database of NCBI Influenza resources[15].

In our experiment, we prepared primitive items as all possible pairs of a position and a amino acid type. We used $20 \times 328 = 6,560$ different items in total. For example, the item "125T" means that

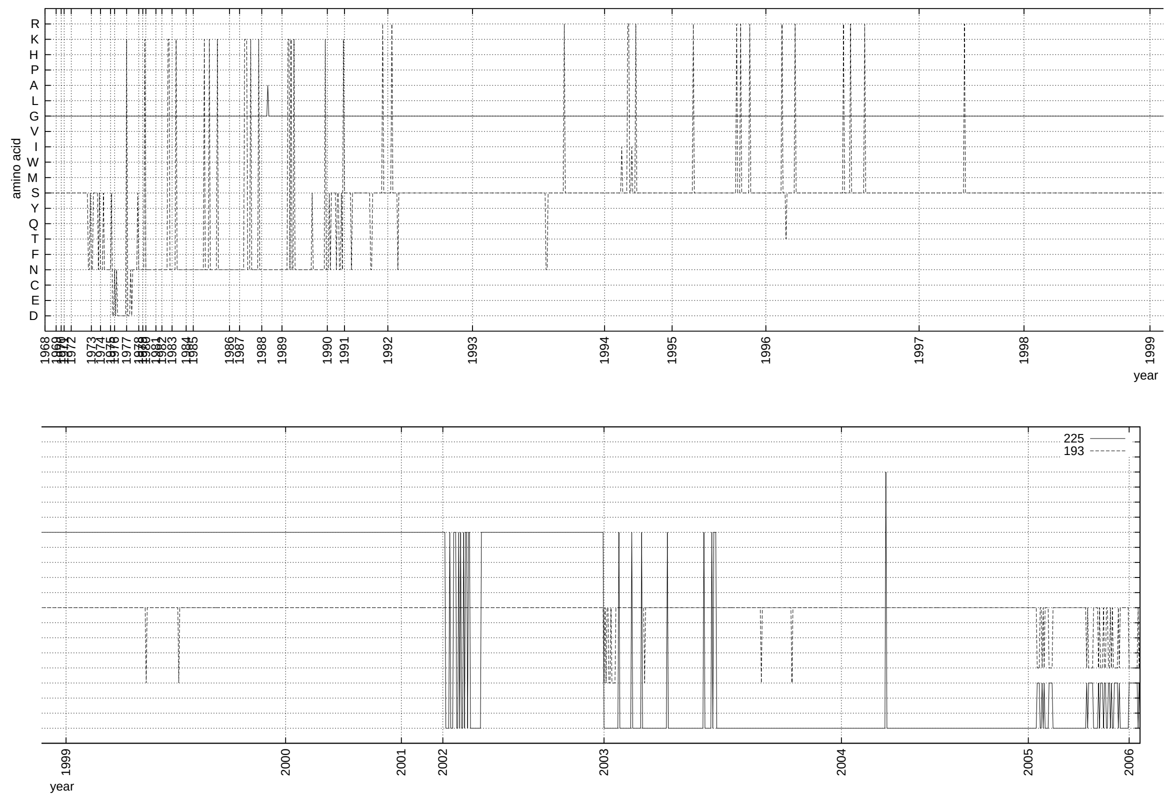


Figure 9: Mutations at the 225th and the 193rd amino acid positions.

the amino acid "T" appears at the 125th position. We used the letter "X" when the data of a position is missing, for instance, "167X" means that we don't know the amino acid at the 167th position. The field of the year is expressed as "Y1989." In this way, the sequence data can be represented as follows:

```
Y1968 1Q 2D 3L 4P 5G ... 326K 327Q 328T
Y1973 1Q 2D 3F 4P 5G ... 326K 327Q 328T
Y1999 1Q 2K 3L 4P 5G ... 326K 327Q 328T
···
```

The data contains 1,657 lines of such combinatorial item sets.

We conducted an experiment of generating a ZBDD for the amino acid sequence data and then extracting all symmetric item sets from the ZBDDs. In our result, the number of ZBDD nodes is 168,261. The CPU time just for ZBDD construction was 17.9 second. Next, we execute our algorithm of symmetric item set mining for the ZBDD. The total computation time for checking 21,513,520(= ${}_{6560}C_2$) item pairs was 725 sec, and we found 64 groups of symmetric items, as follows.

(324P 321R 320M 314L 306P 301T 296N 290N 287S 281C 277C 266S 256G 255R 254P 250N 241D 240G 237V 224R 200G 191Q 181G 166V 153W 147F 136S 134G 132Q 125F 123E 119E 116G 113A 111L 100Y 90R 89E 84W 76C 73D 71L 70L 69A 68D 66L 65T 64C 61G 52C 42L 39A 28T 26V 18H) (13W 12R 11Q 10R 9A 7T 6M) (309A 307X 303R 258X 232M 117X 24X) (303X 187M 180X 133X 120C 60X) (212L 210K 209C 108V 107C) (74X 62X 47X 21X 2X) (322Y 311P 310E 276S 86I) (322X 318X 294C 234X 19X) (328L 327K 326I 323E) (258S 238R 164V 1P) (173G 163M 104E 62A) (225A 211X 183Q 6D) (190N 186D 15M) (203A 198P 11T) (235A 216Y 115P) (214V 23R 3M) (327R 127L 110L) (159R 118M 34T) (146R 139F 31V) (180W 120F 60D) **(225N 193F)** (225H 33R) (104G 93S) (298D 124R) (252T 152S) (218X 140I) (95C 94T) (131D 33N) (87I 80L) (275E 46L) (304V 72X) (140T 139X) (328P 325X) (257F 98N) (219T 2R) (327L 325Q) (193T 183P) (218A 205C) (21R 17R) (144A 126S) (328S 294Y) (187K 156S)

(311E 170H) (244M 83R) (53S 6K) (114L 9K) (309G 213R) (173R 124E) (212A 133G) (172A 80R) (292Q 291H) (15X 14X) (97S 96Y) (236T 226P) (319R 249A) (278V 169S) (319G 249G) (318T 234W) (257Y 98Y) (235T 115S) (232I 24T) (210Q 108L) (127W 110S) (126T 83T)

In the result, the first symmetric group of a large number of items represents the items commonly appear in all the sequences, in other words, they are the amino acid positions which have never changed. The other symmetric groups are related to a part of sequences, and the pair of positions may have some interesting biological relationship. For example, (225N 193F) is one of the symmetric item pair, and we checked the mutations of the 225th and the 193rd amino acid positions with our visualization tool[5]. Some portions of the graphic output are shown in Fig. 9. We can observe that the two amino acid positions have a strong co-relation in the very recent sequences after 2005.

There is a related work[6] to extract co-related amino acid positions based on mutual information analysis. The calculation of mutual information also gives a relationship between the two positions, however, it indicates the total behaviors of all sequence data, and would not be effective to find a relationship sharply seen in a portion of sequence instances. Our method will be useful to detect such relationships.

The symmetric item sets extracted in our method does not always correspond to biologically meaningful relationships, however, some of them may have such interesting information. We have already known many kind of data mining techniques such as frequent pattern mining, closed pattern mining, etc. Our symmetric item set mining will be a new alternative method for knowledge discovery.

6 Conclusion

In this paper, we presented an efficient method for extracting all symmetric item sets in transaction databases. The experimental results show that our method efficiently extracts hidden information from the large-scale database. It is applicable to a real-life biological database, which includes 6,560 items and requires 21,513,520 pairs of symmetry checking. Our method will be useful to detect a sharp relationship hidden in a limited portion of database and may also be useful for pruning noisy data.

As our future work, we are considering more efficient algorithm to be applied for more larger ZBDDs, and it would also be interesting to develop "approximately" symmetry checking method which allows some errors or noise in the data.

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, edtors, *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*, Vol. 22(2) of SIGMOD Record, pp. 207–216, ACM Press, 1993.

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

[3] B. Goethals, "Survey on Frequent Pattern Mining", Manuscript, 2003. `http://www.cs.helsinki.fi/u/goethals/publications/survey.ps`

[4] B. Goethals, M. Javeed Zaki (Eds.), Frequent Itemset Mining Dataset Repository, Frequent Itemset Mining Implementations (FIMI'03), 2003. `http://fimi.cs.helsinki.fi/data/`

[5] Hokkaido University Research Center for Zoonosis Control, PlotH3N2, Available from: `http://www.czc.hokudai.ac.jp/~murakami/plot-H3N2/`

[6] K. Ito, M. Igarashi, H. Kida, and A. Takada, "Computer Analysis of Structural Changes in H3 Hemagglutinins of Human Influenza Viruses Isolated During 1968 to 2006," Thirteenth International Conference Negative Strand Viruses 2006, p.28, 2006.

[7] N. Kettle and A. King: "An Anytime Symmetry Detection Algorithm for ROBDDs," In Proc. IEEE/ACM 11th Asia and South Pacific Design Automation Conference (ASPDAC-2006), pp. 243–248, Jan. 2006.

[8] Krug, R.M. and Lamb, R.A.,Orthomyxoviridae: The Viruses and Their Replication, *Fields Virology. 4th edition*, editors: Knipe DM, Howley PM,. Philadelphia: Lippincott Williams & Wilkins. 2001.

[9] Minato, S., Zero-suppressed BDDs for set manipulation in combinatorial problems, In Proc. 30th ACM/IEEE Design Automation Conf. (DAC-93), (1993), 272–277.

[10] S. Minato: "Binary Decision Diagrams and Applications for VLSI CAD", Kluwer Academic Publishers, November 1996.

[11] S. Minato, Zero-suppressed BDDs and Their Applications, International Journal on Software Tools for Technology Transfer (STTT), Springer, Vol. 3, No. 2, pp. 156–170, May 2001.

[12] S. Minato and H. Arimura: "Efficient Combinatorial Item Set Analysis Based on Zero-Suppressed BDDs", IEEE/IEICE/IPSJ International Workshop on Challenges in Web Information Retrieval and Integration (WIRI-2005), pp. 3–10, Apr., 2005.

[13] S. Minato: "Finding Simple Disjoint Decompositions in Frequent Itemset Data Using Zero-suppressed BDD," In Proc. of IEEE ICDM 2005 workshop on Computational Intelligence in Data Mining, pp. 3-11, ISBN-0-9738918-5-8, Nov. 2005.

[14] A. Mishchenko, "Fast Computation of Symmetries in Boolean Functions," IEEE Trans. Computer-Aided Design, Vol. 22, No. 11, pp. 1588–1593, 2003.

[15] National Center for Biotechnology Information, "Influenza virus resource," Jun 23, 2005. Available from: `http://www.ncbi.nlm.nih.gov/genomes/FLU/FLU.html`

[16] T. Uno, T. Asai, Y. Uchida and H. Arimura, "An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases," In Proc. of the 8th International Conference on Discovery Science 2004 (DS-2004), 2004.