TCS-TR-B-10-7

TCS Technical Report

Recent and Future Work on Decision Diagrams and Discrete Structure Manipulation

by

Shin-ichi Minato

Division of Computer Science

Report Series B

July 2, 2010

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

Recent and Future Work on Decision Diagrams and Discrete Structure Manipulation

Shin-ichi Minato*
Division of Computer Science
Hokkaido University
Sapporo 060-0814, Japan

July 2, 2010

(Abstract) Binary Decision Diagram (BDD) is an efficient data structure for representing Boolean functions. Techniques of BDD manipulation have been developed in the area of VLSI logic design since 1990's. We found that BDD-based techniques can also be applied effectively to problems of data mining and knowledge discovery. Especially, Zero-suppressed BDDs (ZDDs), a type of BDDs, are suitable for handling sets of sparse combinations that often appear in many problems of database analysis. Recently, we have been working on ZDD-based large-scale data processing for such applications.

In most of our research work, we can observe that discrete structure manipulation is a key technique to solve many kinds of real-life problems. In this article, we discuss the basic techniques of discrete structure manipulation using BDDs, ZDDs, and more extended decision diagrams. We also show some interesting applications of those decision diagram-based techniques. Finally we present a plan of our new research project on discrete structure manipulation systems.

1 Introduction

Binary Decision Diagrams (BDDs) [2] are an efficient data structure for representing Boolean functions. Since the 1990's, wide use of BDDs has occurred in VLSI logic design. Recently, we found that BDD-based techniques can also be applied effectively to problems in data mining and knowledge discovery. Especially, Zero-suppressed BDDs (ZDDs) [8] are suitable for handling sets of sparse combinations that often appear in many practical database analyses. Recently, we have been working on ZDD-based large-scale data processing for such applications.

In most of our research work, we can observe that discrete structure manipulation is a key technique to solve many kind of real-life problems. Discrete structures are foundational material for computer science and mathematics, which are related to set theory, symbolic logic, inductive proof, graph theory, combinatorics, probability theory, etc. Many problems solved by computers can be decomposed as a type of discrete structures using simple primitive algebraic operations. It is very important to compactly represent discrete structures and to efficiently execute the tasks such as equivalency/validity checking, analysis of models, optimization, etc. We can see that BDDs and ZDDs are one of the basic data structures for various models of discrete structure manipulation.

In this article, we discuss the basic techniques for manipulating BDDs, ZDDs, and more extended decision diagrams. We also show some interesting applications of those decision diagram-based techniques. Finally we present a plan of our new research project on discrete structure manipulation systems, accepted by Japan Science and Technology Agency as an "ERATO" (Exploratory Research for Advanced Technology) project.

*He also works for ERATO MINATO Discrete Structure Manipulation System Project, Japan Science and Technology Agency.

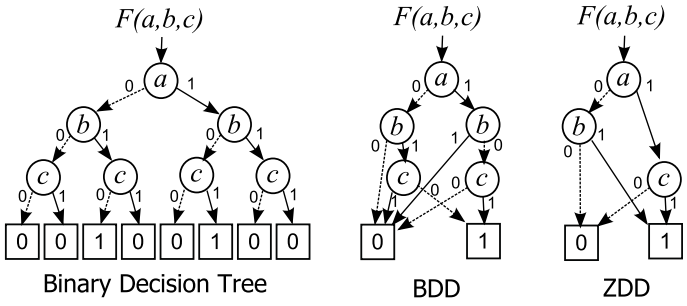

Figure 1: Binary Decision Tree, BDDs and ZDDs

a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$\rightarrow S$

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

$\rightarrow b$

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

$\rightarrow ac$

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

2 BDDs/ZDDs for Discrete Structure Manipulation

2.1 BDDs

A *Binary Decision Diagram (BDD)* is a graph representation for a Boolean function. An example is shown in Fig. 1 for $F(a,b,c) = a\overline{b}c \vee \overline{a}b\overline{c}$. It is derived by reducing a binary decision tree graph, which represents a decision making process by the input variables. If we fix the order of input variables (a, b, c in our example), and apply the following two reduction rules, then we have a compact canonical form for a given Boolean function:

(1) Delete all redundant nodes whose two edges have the same destination, and

(2) Share all equivalent nodes having the same child nodes and the same variable.

The compression ratio by using a BDD depends on the property of Boolean function to be represented, but it can be 10 to 100 times in some practical cases. In addition, using Bryant's algorithm [2], we can systematically construct a (compressed) BDD as the result of a binary logic operation (i.e. AND, OR) for a given pair of (compressed) BDDs, without generating uncompressed binary decision trees. This algorithm is based on hash table techniques, and computation time is almost linear to the BDD size.

BDD is based on the in-memory data processing techniques, and it enjoys the advantage of using random access memories. Recently, commodity PCs are equipped with Giga-Bytes of main memory, and we can solve larger-scale problems which used to be impossible due to memory shortage. Thus, especially after 2000s, the BDD application areas are spreading.

2.2 ZDDs

BDDs were originally invented to represent Boolean functions. However, we can also map a set of combinations into the Boolean space of n variables, where n is the cardinality of the combinatorial items (Fig. 2). So, we could also use BDDs to represent sets of combinations. When

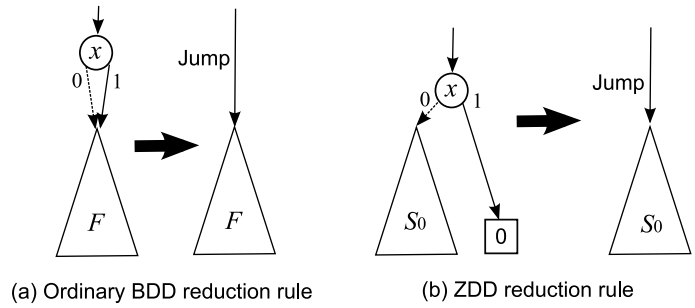

Figure 3: ZDD reduction rule.

Table 1: Primitive ZDD operations

$\emptyset$	Returns empty set. (0-termial node)
$\{\lambda\}$	Returns the set of only null-combination. (1-terminal node)
$P.top$	Returns item-ID at root node of P.
$P.offset(v)$	Subset of combinations not including item v.
$P.onset(v)$	Gets $P \setminus P.offset(v)$ and then deletes v from each combination.
$P.change(v)$	Inverts existence of v (add / delete) on each combination.
$P \cup Q$	Returns union set.
$P \cap Q$	Returns intersection set.
$P \setminus Q$	Returns difference set. (in P but not in Q.)
$P.count$	Counts number of combinations.

(Extended operations introduced by Minato [9])

$P * Q$	Cartesian product of P and Q.
P/Q	Quotient of P divided by Q.
$P\%Q$	Reminder of P divided by Q.

a set of combinations consists of many similar sub-combinations, the BDD gives significant data compression effect and the manipulation becomes faster. In this way, BDDs are applied not only for VLSI logic design but also for various combinatorial problems such as data mining, constraint satisfaction problems, fault analysis of large systems, etc.

Zero-suppressed BDD (ZDD, or ZBDD) [8] is a variant of BDD, customized for manipulating sets of combinations. ZDDs are based on the special reduction rules different from ordinary ones. As shown in Fig. 3, all of the nodes whose 1-edge directly points to the 0-terminal node are deleted. As the result, the nodes of items that do not appear in any sets of combinations are automatically deleted (Fig.1). This ZDD reduction rule is extremely effective for handling a set of sparse combinations. If the average appearance ratio of each item is 1%, ZDDs are possibly more compact than ordinary BDDs, even up to 100 times more. Such situations often appear in real-life problems, for example, in a supermarket, the number of items in a customer's basket is usually much less than all the items displayed there.

ZDD representation has another good property, which is 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 ZDD equals the number of combinations in the set. This attractive property indicates that, even if there are no equivalent nodes to be shared, the ZDD structure explicitly stores all the items of each combination, as well as uses an explicit linear linked list data structure. In other words, (the order of) the size of the ZDD never exceeds the explicit rep-

resentation. If more nodes are shared, the ZDD is more compact than the linear list.

Table 1 summarizes most of the primitive operations of the ZDDs. In these operations, $\emptyset$, $\{\lambda\}$, and $P.top$ can be obtained in a constant time. $P.offset(v)$, $P.onset(v)$, and $P.change(v)$ operations require a constant time if v is the top variable of P, otherwise they consume linear time for the number of ZDD nodes located at a higher position than v. The union, intersection, and difference operations can be performed in almost linear time to the size of the ZDDs.

The last three operations in the table constitute an interesting algebra for sets of combinations with multiplication and division. Knuth has been inspired from this idea and developed more various algebraic operations such as $P \sqcap Q$, $P \sqcup Q$, $P \boxplus Q$, $P \nearrow Q$, $P \searrow Q$, $P^{\uparrow}$, $P^{\downarrow}$, etc. He named those ZDD-based operations "Family Algebra" in the recent fascicle of his famous book series [5].

ZDD is now widely recognized as the most important variant of BDD, and commonly used as well as ordinary BDD.

3 Database Analysis Based on ZDD Manipulation

3.1 Frequent Itemset Mining

Frequent itemset mining is one of the fundamental problems for data mining and knowledge discovery. The task is to find all frequent itemsets that are item-combinations included in at least θ records of the itemset database where θ is the user specified threshold.

Figure 4 shows a small example of frequent itemset mining. Here we use the letters $a, b, c, \ldots$ to represent items, and a set of itemsets is written as $\{ab, bc, a, b, c, \lambda\}$. (Here λ means null itemset.) In this example, the database D has 11 transactions in total. The itemset b occurs 10 times out of 11. The itemsets a, c, and ac occur 8 times, bc occurs 7 times, ac and abc occur 5 times. Thus, if we specify $\theta = 7$, $\{ab, bc, a, b, c, \lambda\}$ is obtained as the set of frequent itemsets. We can observe that when the minimum support θ is smaller, more itemsets become frequent. If $\theta = 1$, any subset of transactions can be frequent, so the number of frequent itemsets can grow exponentially with the maximal number of items in one record.

Since the pioneering work by Agrawal *et al.*[1], various algorithms have been proposed to solve the *frequent itemset mining problem* (cf., [4, 14]). Recently, Minato et al. [12] proposed a fast algorithm "LCM over ZDDs" for generating very large-scale frequent itemsets using *Zero-suppressed BDDs (ZDDs)* [8], a compact graph-based data structure. This method is based on *LCM algorithm* [13], one of the most efficient state-of-the-art techniques for itemset mining, and directly generates compact output data structures on the main memory, to be efficiently post-processed by using ZDD-based algebraic operations.

3.2 Itemset-Histograms and ZDD Vectors

In many applications of itemset data mining, we need to count the frequency (number of occurrence) for each itemset in the given databases. An *itemset-histogram* is a kind of table that lists the frequency of all itemsets. Figure 5 shows examples of itemset-histogram for DB records in Fig. 4, and one for counting all sub-patterns occurred in the databases. These histograms sometimes grow very large, so efficient data structure is a key technique in itemset mining problems.

Because ZDDs are representations of sets of combinations, a simple ZDD distinguishes only the existence of each itemset in the database. In order to represent the frequency number of each itemset, we decompose the number into the m digits of a "ZDD vector" $\{F_0, F_1, \ldots, F_{m-1}\}$ to represent integers up to $(2^m - 1)$, as shown in Fig. 6. That is, we encode the frequency numbers into the binary digits, with F_0 representing the itemsets occurs an odd number of times (LSB = 1), F_1 representing the itemsets whose frequency number's second lowest bit is 1, and similarly for each digit up to F_{m-1}.

In the example of Fig. 6, the frequencies of itemsets are decomposed as: $F_0 = \{abc, ab, c\}$, $F_1 = \{ab, bc\}$, $F_2 = \{abc\}$, following which each

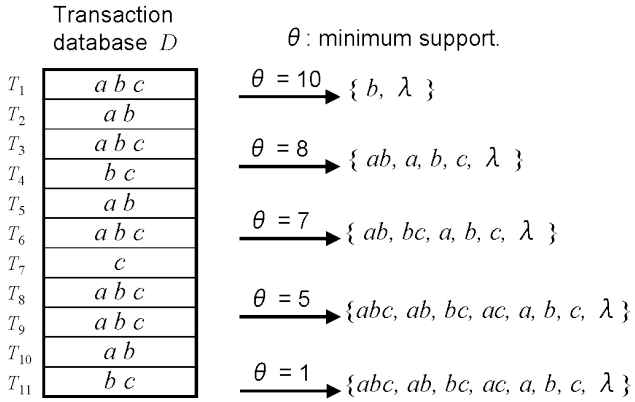

Figure 4: Frequent itemset mining.

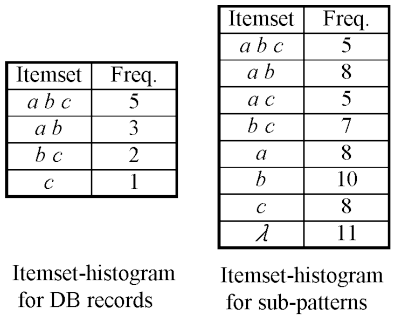

Figure 5: Example of itemset-histogram.

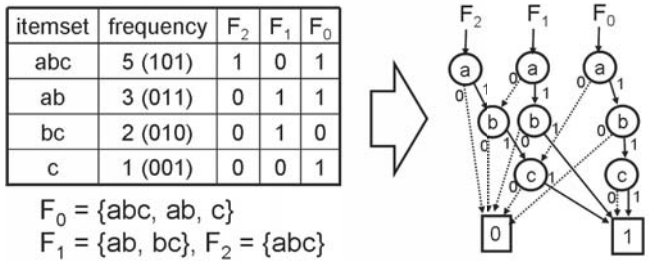

Figure 6: ZDD vector for itemset-histogram.

digit can be represented by a simple ZDD. The three ZDDs share their subgraphs with each other.

When we construct a ZDD vector for an itemset-histogram, the number of ZDD nodes in each digit is bounded by the total occurrences of items in all itemsets. If there are many partially similar itemsets in the database, the subgraphs of ZDDs will be well shared, and a compact representation is obtained. The bit-width of the ZDD vector is bounded by $\log S_{max}$, where S_{max} is the appearance of the most frequent items.

3.3 Itemset-Histogram Algebra

Once we have generated a ZDD vector for the itemset-histogram, various operations can be executed efficiently. Table 2 lists primitive algebraic operations for itemset-histograms. These operations can be composed as a sequence of ZDD operations. The operation result is also compactly represented by a ZDD vector. The computation time bound is approximately linearly related to the total ZDD size.

Composing of some sequences of those algebraic operations, we can apply various post-processing for data analysis after frequent itemset mining, such as [11]:
- sub-pattern matching for frequent itemsets,
- extracting long/short patterns,
- comparison between two sets of frequent itemsets, and
- calculating statistical data. (support, confidence, etc.)

From Table 1 and 2, we can observe that the two sets of algebraic operations have strong correspondences to each other. This means that itemset-histogram algebra is a natural extension of one for the simple sets of combinations. It is reasonable because an itemset-histogram can be regarded as a "multiset of combinations." For multiset of combinations, it is natural to restrict the occurrence number of each combination to be a positive integer, but we may more extend it to allow negative numbers. We named such data model "Valued

Table 2: Algebraic operations for itemset-histgrams.

c $(c \in N)$	Returns the histogram of only λ with occurrence c. ($c = 0$ means empty.)
$P.top$	Returns item-ID of the highest order involved in P.
$P.offset(v)$	Returns sub-histogram of itemsets not including item v.
$P.onset(v)$	Gets $P - P.offset(v)$ and then deletes v from each itemset.
$P.change(v)$	Inverts existence of v (add / delete) on each itemset.
$Max(P,Q)$	Histogram with the greater occurrences between P and Q.
$Min(P,Q)$	Histogram with the less occurrences between P and Q.
$P - Q$	Difference of the two histograms. (limited to only positive occurrences.)
$P.count$	Counts the variety of itemsets in P.
$P.upperbound$	Returns the highest occurrence number among all itemsets in P.
$P.unitset$	Returns the histogram with occurrence = 1 for all itemsets in P.
$P * Q$	Cartesian product of P and Q.
P/Q	Quotient of P divided by Q.
$P\%Q$	Reminder of P divided by Q.

Sum-Of-Products" (VSOP) expression, and we developed a C-Shell-like script interpreter to easily describe VSOP expressions and efficiently calculate them using ZDD vectors [10]. This program is opened for free, and anyone can try it.

4 Sequence BDDs for Sets of Sequences

4.1 Representation for Sets of Sequences

A *set of sequences* (or set of strings) is a very popular model for representing various data such as text documents, gene sequences, sequential events, etc. This model is also called a *language* in traditional computation theory. $\{aaa, aba, bbc, bc\}$ is an example of set of sequences where the set of symbols $\Sigma = \{a, b, c\}$. Here we consider only finite sets of sequences.

Since ordinary ZDDs represent sets of combinations, the order of symbols (e.g. $\{ab, ba\}$) and duplicated symbols (e.g. $\{aa, aaa, ab, aabb\}$) cannot be distinguished. A naive solution to handle sets of sequences by ZDDs is using encoded symbols to specify the positions in the sequences [6]. For example, the sets of sequences $\{aaa, aba, bbc, bc\}$ is encoded as $\{a_1a_2a_3, a_1a_2a_3, b_1b_2c_3, b_1c_2\}$. Figure 7 shows an example of ZDD using such encoding. Here a_1, a_2, a_3 are the different symbols for the same letter but different positions, and thus it can be regarded as a set of combinations.

In this technique, we can represent and manipulate the sets of sequences by ZDD-based algebraic operations. However, the number of symbols increases as many times as the length of the longest sequence. There is another inconvenient point that we need to fix the maximum length of the sequences in the beginning of ZDD construction. Sometimes it is hard to estimate.

4.2 Sequence BDDs

Recently, Loekito, Bailey, and Pei [7] proposed *Sequence BDD (SeqBDD)*, which is a new ZDD-based data structure for representing sets of sequences. This is almost same as ZDD but only differs in variable ordering rule. SeqBDDs applies a fixed variable order constraint only to the 0-edge, but the 1-edge side has no ordering constraint. As shown in Fig. 8, this harf-relaxed rule allows a same letter to occur multiple times in a path. Figure 9 shows an example of SeqBDD. In the SeqBDD semantics, each path in a SeqBDD represents a sequence, for which the nodes are arranged in order of the positions of their respective variables in the sequence. More specifically, the top node corresponds to the head of the sequence, and the successive nodes take 1-edges corresponding to the following letters, respectively.

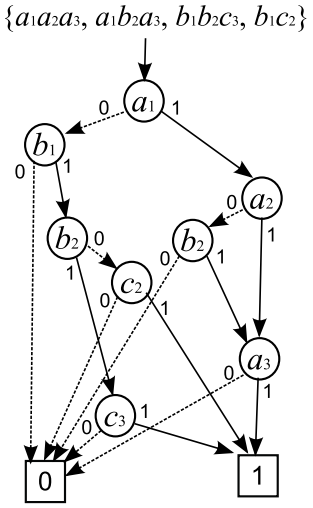

Figure 7: Am encoded ZDD for sets of sequences.

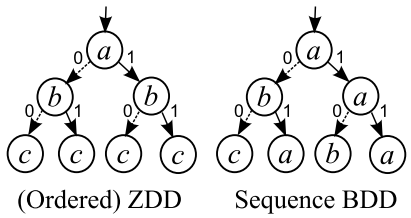

Figure 8: Variable ordering rules.

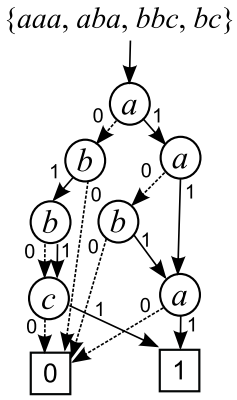

Figure 9: A SeqBDD.

Table 3 summarizes primitive algebraic operations of SeqBDDs. We can see that this is very close to ZDD algebra. *onset, offset,* and *push* operations are slightly different from ordinary ZDDs. In principle, the SeqBDDs have a special meaning at the first position of the sequences. Each decision node checks the first letter of all sequences, and then divides them into two sub-sequences. However, other binary operations such as union, intersection, difference, etc. are almost similar. Another interesting point is why SeqBDD is an extension of ZDD. We can see that zero-suppressed reduction rule is necessary to SeqBDD reduction. Ordinary (symmetric) BDD reduction rule is not applicable because it conflicts with asymmetric variable ordering rule in SeqBDDs.

SeqBDDs is useful since they can directly represent sets of sequences without position-based symbol encoding, so we do not have to know the maximum length of sequences at the beginning. Of course the sequence length should be a finite number in SeqBDDs but they have no fixed upper bound. Only the total number of decision nodes is limited by the main memory capacity. SeqBDDs are efficient especially for representing sets of sequences containing very long sequences and some large number of short sequences. Denzumi et al.[3] presented a technique "Suffix-DD," which is a suffix-tree manipulation based on SeqBDDs. This method efficiently constructs an index of all sub-

Table 3: Algebraic operations for SeqBDDs

$\emptyset$	Returns empty set. (0-termial node)
$\{\lambda\}$	Returns the set of only null-combination. (1-terminal node)
$P.top$	Returns item-ID at root node of P.
$P.onset(x)$	Selects the subset of sequences that begin with letter x, and then removes x from the head of each sequence.
$P.offset(x)$	Selects the subset of sequences that do not begin with letter x.
$P.push(x)$	Appends x to the head of every sequence in P.
$P \cup Q$	Returns union set.
$P \cap Q$	Returns intersection set.
$P \setminus Q$	Returns difference set. (in P but not in Q.)
$P.count$	Counts number of combinations.
$P * Q$	Cartesian product of P and Q. (Concatenations of all pairs in P and in Q)

strings for a given long text string. The techniques of suffix-trees are commonly used in many practical applications, so we hope that many of those applications will be accelerated by our new data structures and algorithms.

5 Discrete Structure Manipulation System Project

Since there are so many interesting and useful ideas on decision diagrams and discrete manipulation, we considered this is a chance to start a research project to focus on BDDs/ZDDs and newly organize their algebra. Fortunately, our proposal has been accepted by JST (Japan Science and Technology Agency) as an ERATO (Exploratory Research for Advanced Technology) project, one of the prestigious projects in Japan. Our project has been authorized in October 2009. The main research activity is started from April 2010 and continues for five years.

Figure 10 illustrates the technical areas of this project. We have a layer of the computation theory as a basis of computer science, which is commonly used for various problems. On the other hand, there are a number of application-specific technical areas in parallel, customized for solving real-life problems. As the middle layer of the two, we have our subject, "discrete structure manipulation system." This layer does not have clear borders between the upper or lower layers, but the research principle is distinctive. Here we respect the conceptual or theoretical results, but put more importance on implementation of the algorithms for practical use. In addition, we also respect the beauty of simplicity and universality, to avoid an ad hoc solution for each application-specific problem. We believe that this research principle features that "this is Art," not just Science or Engineering. In this project, we place BDDs/ZDDs as a set of basic techniques for manipulating various types of discrete structures. The objectives of this project are as follows.

(1) Based on the primitive logic and set family operations, we will organize algebraic operations for manipulating more general discrete structures, and apply it to various technical areas, and

(2) We will also provide implementation techniques of an efficient and robust manipulation system for domestic and international researchers as well as for industry, in a proper format to be easily utilized.

For (1), we will extensively study the previous techniques on typical discrete structure models, and will newly organize the efficient algorithms based on BDDs/ZDDs. We also study various application-specific problems and analyze their structures, aiming to construct a general system of manipulating higher-level structures. Especially for the traditional research area (e.g. statistical

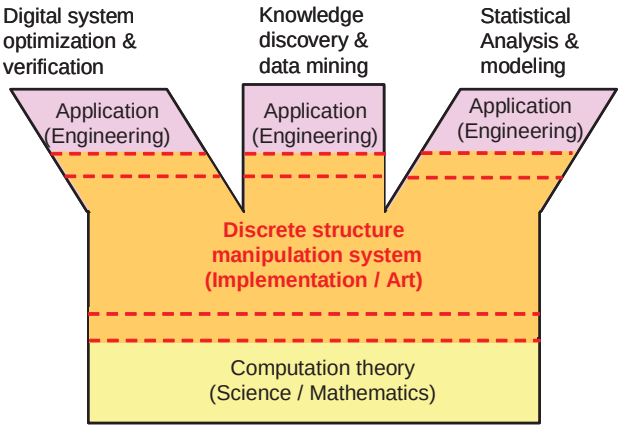

Figure 10: Technical Areas of our ERATO project.

analysis), our new approach would have an impact to give further improvement to the current technology. Those research results are mainly returned to the society by publication of papers and textbooks.

For (2), our software products developed in this project requires not only high performance, but also good support service for users. Currently, several academic groups provide open software of BDD packages, however, in many cases, the source code was written by a teaching staff and he (or she) does not enough time to answer user's inquiries and code maintenance. Students may have more time than professors, but students are not stable members, and they cannot write a thesis only by the code maintenance. To solve this problem, a part of our ERATO project members, loaned from an enterprise or with a concurrent post, will extensively study the implementation techniques of our software systems. Future they will bring the techniques back to their original organization and utilize them for industrial applications. This is a way of contribution to the society, as "technology transfer by persons."

6 Conclusion

In this article, we presented several recent topics on decision diagram-based data structure and algebraic operations: BDDs for Boolean functions, ZDDs for sets of combinations, ZDD vectors for itemset-histgrams, and SeqBDDs for sets of sequences. They have somehow similar algebraic structures, and strongly related to each other. Each of them has interesting properties and will be useful for various practical applications.

In most of our research work, discrete structure manipulation is a key technique to solve many kinds of real-life problems. We planned a new research project to focus on BDDs/ZDDs and newly organize their algebra. Our research activity is started from April 2010 and continues for five years. Excellent researchers who are interested in discrete structure manipulation are welcome to our project.

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] Shuhei Denzumi, Hiroki Arimura, and Shin ichi Minato. Substring indices based on sequence bdds. *Hokkaido University, Division*

of Computer Science, TCS Technical Reports, TCS-TR-A-10-42, 2010.

[4] B. Goethals. Survey on frequent pattern mining, 2003. http://www.cs.helsinki.fi/u/goethals /publications/survey.ps.

[5] D. E. Knuth. *The Art of Computer Programming: Bitwise Tricks & Techniques; Binary Decision Diagrams*, volume 4, fascicle 1. Addison-Wesley, 2009.

[6] R. Kurai, S. Minato, and T. Zeugmann. N-gram analysis based on zero-suppressed BDDs. In *In T. Washio, et al. editors, New Frontiers in Artificial Intelligence, Joint JSAI 2006 Workshop Post-Proceedings, LNAI 4384*, pages 289–300, 2 2006.

[7] E. Loekito, J. Bailey, and J. Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems*, (DOI:10.1007/s10115-009-0252-9), 2009.

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

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

[10] S. Minato. VSOP (Valued-Sum-Of-Products) calculator for knowledge processing based on zero-suppressed BDDs. In *In K. P. Jantke, et al. editors, Federation over the Web, LNAI 3847*, pages 40–58, 2 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, T. Uno, and H. Arimura. LCM over ZBDDs: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. In *Proc. of 12-th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2008), (LNAI 5012, Springer)*, pages 234–246, 5 2008.

[13] 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/.

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