INVITED SURVEY PAPER

Techniques of BDD/ZDD: Brief History and Recent Activity

Shin-ichi MINATO[†*a)], *Senior Member*

SUMMARY 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 into discrete structures using simple primitive algebraic operations. It is very important to represent discrete structures compactly and to execute efficiently tasks such as equivalency/validity checking, analysis of models, and optimization. Recently, BDDs (Binary Decision Diagrams) and ZDDs (Zero-suppressed BDDs) have attracted a great deal of attention, because they efficiently represent and manipulate large-scale combinational logic data, which are the basic discrete structures in various fields of application. Although a quarter of a century has passed since Bryant's first idea, there are still a lot of interesting and exciting research topics related to BDD and ZDD. BDD/ZDD is based on in-memory data processing techniques, and it enjoys the advantage of using random access memory. Recent commodity PCs are equipped with gigabytes of main memory, and we can now solve large-scale problems which used to be impossible due to memory shortage. Thus, especially since 2000, the scope of BDD/ZDD methods has increased. This survey paper describes the history of, and recent research activity pertaining to, techniques related to BDD and ZDD.

key words: *BDD, ZDD, decision diagram, discrete structure, algorithm, data structure*

1. Introduction

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 into discrete structures using simple primitive algebraic operations. It is very important to represent discrete structures compactly and to execute efficiently tasks such as equivalency/validity checking, analysis of models, and optimization. Those techniques are commonly used in many application areas in computer science, for example, hardware/ software system design, fault analysis of large-scale systems, constraint satisfaction problems, data mining, knowledge discovery, machine learning/classification, bioinformatics, and web data analysis. They have considerable ripple effects on today's society.

A Binary Decision Diagram (BDD) [1] is a representation of a Boolean function, one of the most basic models of discrete structures. Systematic methods for Boolean function manipulation were first studied by Shannon in 1938, who applied Boolean algebra to logic network design. The AND-OR two-level structure (called DNF or CNF) has been used for a long time as the data structure of Boolean functions. However, after the epoch-making paper by R.E. Bryant in 1986, BDD-based methods have become a hot topic and have been rapidly developed.

BDD was originally invented for the efficient Boolean function manipulation required in VLSI logic design, but Boolean functions are also used for modeling many kinds of combinatorial problems. Zero-suppressed BDD (ZDD) [2] is a variant of BDD, customized for manipulating "sets of combinations." ZDDs have been successfully applied not only for VLSI design but also for solving various combinatorial problems, such as constraint satisfaction, frequent pattern mining, and graph enumeration. Recently, ZDD has become more widely known, since D.E. Knuth intensively discussed ZDD-based algorithms in the latest volume of his famous series of books [3].

Although a quarter of a century has passed since Bryant first put forth his idea, there are still many interesting and exciting research topics related to BDD and ZDD. For example, Knuth presented a surprisingly fast algorithm "Simpath" [3] to construct a ZDD which represents all the paths connecting two points in a given graph structure. This work is important because many kinds of practical problems are efficiently solved by some variations of this algorithm. Another example of recent activity is to extend BDDs to represent other kinds of discrete structures, such as sequences and permutations. In this context, new variants of BDDs called sequence BDD [4] and πDD [5] have recently been proposed and the scope of BDD-based techniques is now increasing.

This survey paper describes the history of, and recent research activity into, the techniques related to BDD and ZDD. We beginning by explaining the basic techniques for BDD/ZDD manipulation in Sect. 2. Next we give a brief history of BDD/ZDD-related work in Sect. 3, and then some recent activity is discussed in Sect. 4. Section 5 concludes this article.

2. Basic Techniques for BDD/ZDD Manipulation

In this section, we describe the basic data structures and algorithms for manipulating BDDs and ZDDs.

Manuscript received November 28, 2012.
Manuscript revised March 21, 2013.

[†]The author is with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.
[*]The author is also with JST ERATO MINATO Discrete Structure Manipulation System Project.
a) E-mail: minato@ist.hokudai.ac.jp

DOI: 10.1587/transinf.E96.D.1419

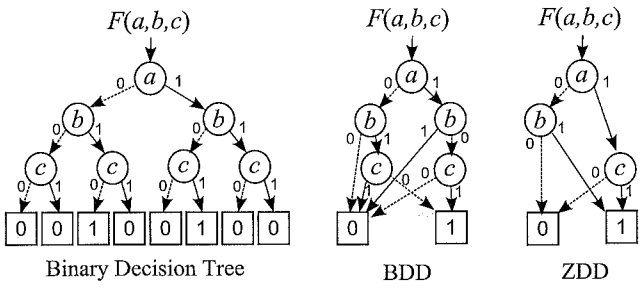


Fig. 1 Binary Decision Tree, BDDs and ZDDs.

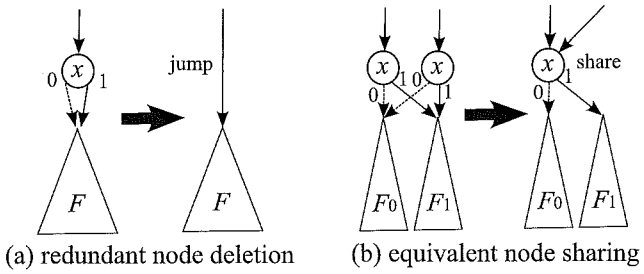


Fig. 2 BDD reduction rules.

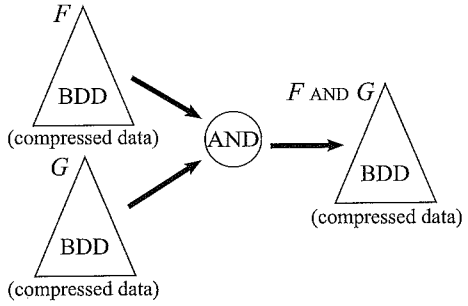


Fig. 3 Framework of BDD logic operation.

Table 1 Primitive BDD operations.

$\emptyset$	Returns the constant-0 function. (0-terminal node)
1	Returns the constant-1 function. (1-terminal node)
F.top	Returns the variable-ID at the root node of F.
$F_{(x=0)}$	Returns a subfunction of F when variable $x = 0$.
$F_{(x=1)}$	Returns a subfunction of F when variable $x = 1$.
$\overline{F}$	Logical NOT operation.
$F \wedge G$	Logical AND operation.
$F \vee G$	Logical OR operation.
$F \oplus G$	Exclusive-OR operation.
F.count	Returns the number of input assignments that satisfy $F = 1$.

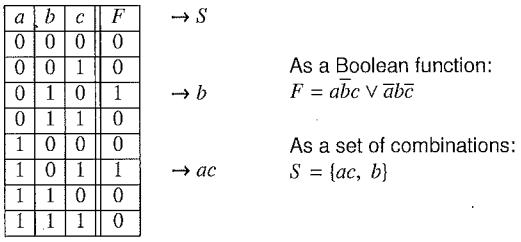


Fig. 4 Correspondence of Boolean functions and sets of combinations.

2.1 BDDs

A Binary Decision Diagram (BDD) is a graphical representation of a Boolean function, which was originally developed for VLSI design. As illustrated in Fig. 1, it is derived by reducing a binary decision tree, which represents a decision making process that depends on some input variables. In this graph, we may find the following two types of decision nodes:

(a) *Redundant node*: A decision node whose two child nodes are identical.
(b) *Equivalent nodes*: Two or more decision nodes having the same variable and the same pair of child nodes.

If we find such types of nodes, we can reduce the graph without changing the semantics (in other words, we can compress the graph) based on the rules shown in Fig. 2 (a)(b). If we fix the order of input variables and apply the two reduction rules as much as possible, then we obtain a canonical form for a given Boolean function [6]. Such a data structure is called an Ordered BDD (OBDD), but in this article we will just call it a BDD.

The compression ratio of a BDD depends on the properties of Boolean function to be represented, but it can be 10 to 100 times more compact in some practical cases. In addition, we can systematically construct a BDD that is the result of a binary logic operation (i.e., AND or OR) for a given pair of BDDs, as shown in Fig. 3. This algorithm, proposed by Bryant [1], is based on a recursive procedure with hash table techniques, and it is much more efficient than generating binary decision trees when the BDDs have a good compression ratio. The computation time is bounded by the product of the BDD sizes of the two operands, and in many practical cases, it is linearly bounded by the sum of input and output BDD sizes†. Table 1 summarizes the list of primitive BDD operations.

BDDs are based on in-memory data processing techniques, and enjoy the advantage of using random access memory. Recent commodity PCs are equipped with gigabytes of main memory, and we can now solve large-scale problems which used to be impossible due to memory shortage. Thus, especially since 2000, the scope of BDD has increased.

2.2 ZDDs

BDDs were originally invented for Boolean function manipulation. But we can also map a set of combinations into the Boolean space of n variables, where n is the cardinality of the set of combinatorial items, as shown in Fig. 4. So, one could also use a BDD to represent a set of combinations. When a set consists of many similar combinations, the BDD

†Recently, the counterexamples have been given for some special instances [7], but in many practical cases, computation time is linearly bounded by input and output BDD sizes.

provides a significant data compression and the manipulation becomes faster. In this way, BDDs can be applied not only to VLSI logic design but also to various combinatorial problems, such as constraint satisfaction problems and fault analysis of large systems.

Zero-suppressed BDDs (ZDDs, or ZBDDs) are a variant of BDDs, customized to manipulate sets of combinations. An example is shown in Fig. 1. This data structure was first introduced in 1993 by Minato [2]. ZDDs are based on special reduction rules that differ from the ordinary ones. As shown in Fig. 5, we delete all nodes whose 1-edge directly points to the 0-terminal node, but do not delete the nodes which would be deleted in an ordinary BDD. This new reduction rule is extremely effective if we are handling a set of sparse combinations. If the average appearance ratio of each item is 1%, ZDDs are possibly up to 100 times more compact than ordinary BDDs. 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 the number of all the items displayed.

A ZDD representation has another good property: each path from the root node to the 1-terminal node corresponds to one item 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 at least as compactly as an explicit linear linked list data structure. In other words, (the order of) the size of the ZDD never exceeds that of the explicit representation. If more nodes are shared, the ZDD is more compact than the corresponding linear list.

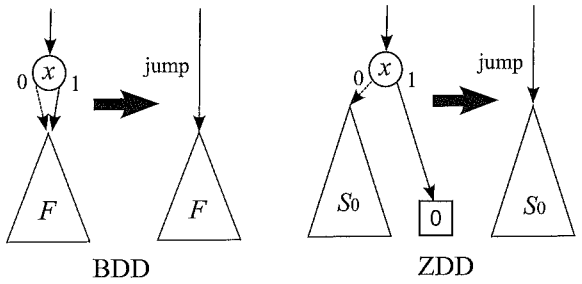


Fig. 5 ZDD reduction rule.

Table 2 summarizes most of the primitive operations of the ZDDs. In these operations, $\emptyset$, $\{\lambda\}$, and *P.top* can be obtained in constant time. Here λ means a null combination. *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 require a time linear in the number of ZDD nodes located at a higher position than v. The union, intersection, and difference operations can be performed in a time that is linear in the size of the ZDDs in many practical cases.

The last three operations in the table constitute an interesting algebra for sets of combinations with multiplication and division. Knuth has been inspired by this idea and has developed more various algebraic operations, such as $P \sqcap Q$, $P \sqcup Q$, $P \boxplus Q$, $P \nearrow Q$, $P \searrow Q$, $P^{\uparrow}$, and $P^{\downarrow}$. He called these ZDD-based operations a "Family Algebra" in the recent fascicle of his book series [3].

The original BDD was invented and developed for VLSI logic design, but ZDD is now recognized as the most important variant of BDD, and is widely used in various kinds of problems in computer science [9]–[12].

3. Brief History of BDD/ZDD Research

Figure 6 gives an overview of research related to BDDs and ZDDs. In this chart, each topic is referred to by author name and year. Of course, it is impossible to cover all the research activities in one chart, so there may be many other interesting topics which are not shown here.

3.1 Overview of BDD/ZDD Related Work

In 1986, Bryant proposed the algorithm called "Apply" [Bryant86][1]. It was the origin of much work on the use of BDDs as modern data structures and algorithms for efficient Boolean function manipulation. Just after Bryant's paper, implementation techniques for the BDD manipulation, such as hash-table operations and memory management techniques, emerged [Brace90][13], [Minato90][14]. A BDD is a unique representation of a given Boolean function. However, if the order of input variables is changed, a

Table 2 Primitive ZDD operations.

$\emptyset$	Returns empty set. (0-terminal node)
$\{\lambda\}$	Returns the set of only null-combinations. (1-terminal node)
P.top	Returns the item-ID at the 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 the existence of v (add / delete) on each combination.
$P \cup Q$	Returns the union set.
$P \cap Q$	Returns the intersection set.
$P \setminus Q$	Returns the difference set. (in P but not in Q.)
P.count	Counts number of combinations.

Extended operations for ZDDs [8]

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

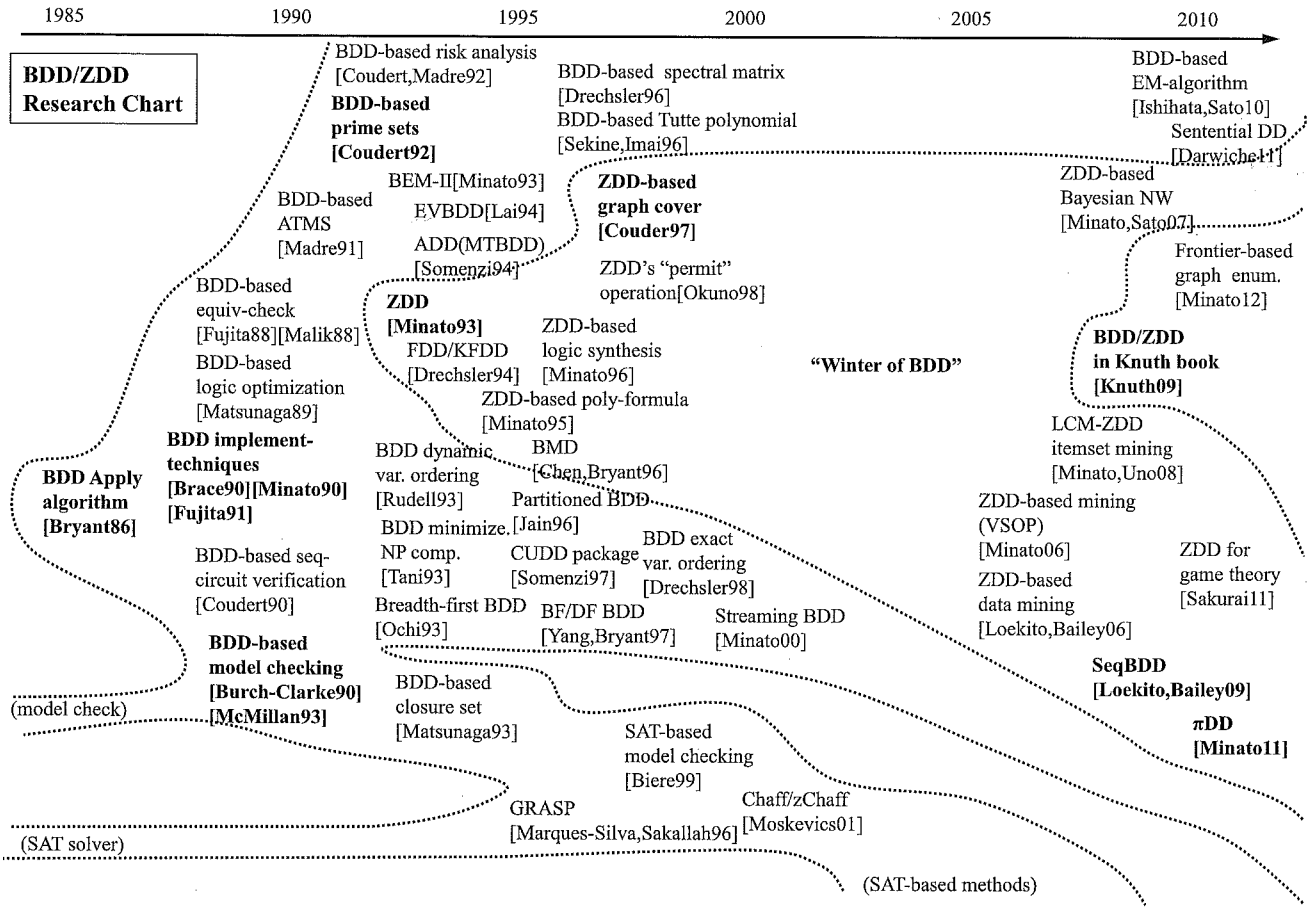


Fig. 6 Chart of BDD/ZDD related work.

different BDD is obtained for what is essentially the same Boolean function. Since the size of BDDs greatly depends on the order of input variables, variable ordering methods were intensively developed in the early days [Fujita91][15], [Rudell93][16], [Tani93][17], [Drechsler98][18]. Those practical techniques were implemented in a software library called "BDD package." Currently, several academic groups provide such packages as open software. ([Somenzi97][19], for example.)

At first, BDDs were applied to equivalence checking of logic circuits [Fujita88][20], [Malik88][21], [Coudert90][22] and logic optimization [Matsunaga89][23] in VLSI logic design. Next, BDD-based symbolic manipulation techniques were combined with the already known theory of model checking. It was really a breakthrough that formal verification becomes possible for some practical sizes of sequential machines [Burch,Clarke90][24], [McMillan93][25]. After that, many researchers became involved with formal hardware verification using BDDs, and Clarke received the Turing Award in 2008 for this work. In addition, the BDD-based symbolic model checking method led to the idea of bounded model checking using SAT solvers [Marques-Silva,Sakallah96][26], [Biere99][27], [Moskewicz01][28]. These research produced many practical applications of SAT solvers that are widely utilized today.

ZDDs [Minato93][2] deal with sets of combinations, representing a model that is different from Boolean functions. However, the original motivation of developing ZDDs was also for VLSI logic design. ZDDs were first used for manipulating very large-scale logic expressions with an AND-OR two-level structure (also called DNF or CNF), namely, representing a set including very large number of combinations of input variables. Sets of combinations often appear not only in VLSI logic design area but also in various areas of computer science. It is known that ZDDs are effective for handling many kinds of constraint satisfaction problems in graph theory and combinatorics [Coudert97][9], [Okuno98][29].

3.2 "Winter of BDD" and After

In this BDD/ZDD research chart, one can observe a gap between 1999 and 2005. Of course there are some publications related to BDDs or ZDDs, but the research activity in this period is obviously less than in previous or later years. We may call this gap the "Winter of BDDs." By that time, most basic implementation techniques had been matured, and BDD applications to VLSI design tools seemed to be almost exhausted. So, many researchers moved from BDD-

related work to other research areas, such as actual VLSI chip design issues or SAT-based problem solving.

However, after about 2005, many people understood that BDDs/ZDDs are useful not only for VLSI logic design but also in various other areas, and then BDD-related research activity was revived. For example, we can see some applications to data mining [Loekito,Bailey06][10], [Minato06][30], [Minato,Uno08][31], Bayesian network and probabilistic inference models [Minato,Sato07][11], [Ishihata,Sato10][12], and game theory [Sakurai11][32]. More recently, new types of BDD variants, which have not been considered before, have been proposed. Sequence BDDs [Loekito,Bailey09][4] represent sets of strings or sequences, and πDDs [Minato11][5] represent sets of permutations.

Synchronizing with this new movement, the BDD section of Knuth's book was published [Knuth09][3]. As Knuth presented the potential for wide-ranging applications of BDDs and ZDDs, these data structures and algorithms were recognized as fundamental techniques for whole fields of information science. In particular, his book includes the "Simpath" algorithm which constructs a ZDD representing all the connecting paths between two points in a given graph structure, and a surprisingly fast program is provided for public use on his web page. Experimental results suggest that this algorithm is not just an exercise but is the most efficient method using current technology. Based on this method, the author's research group is now developing extended and generalized algorithms, called "Frontier-based methods," for efficiently enumerating and indexing[†] various kinds of discrete structures [33]–[35].

As the background of this resurgence and new generation applications, we should note the great progress of the computer hardware system, especially the increase of main memory capacity. Actually, in the early days of using BDDs in 1990's, there was some literature on applications for intelligent information processing. Madre and Coudert proposed a TMS (Truth Maintenance System) using BDDs for automatic logic inference and reasoning [Madre92][36]. A method of probabilistic risk analysis for large industrial plants was also considered [Coudert,Madre92][37]. Coudert also proposed a fast method of constructing BDDs to represent prime sets (minimal support sets) for satisfying Boolean functions [Coudert92][38], which is a basic operation for logic inference. However, at that time, the main memory capacities of high-specification computers were only 10 to 100 megabytes, about 10,000 times smaller than those available today, and thus, only small BDDs could be generated.

In the VLSI design process, the usual approach was that the whole circuit was divided into a number of small submodules, and each submodule was designed individually by hand. So, it was natural that the BDD-based design tools are used for the sufficiently small submodules which could be handled with a limited main memory capacity. With the progress of computer hardware performance, BDD-based methods could gradually be applied to larger submodule. On the other hand, in the applications of data mining or knowledge processing, the input data were stored on a very large hard disks. The processor loaded a small fragment of data from the hard disk into the main memory, and executed some meaningful operations on the data in the main memory, then the processed data was saved with the original data on the hard disk. Such procedures were common, but it was very difficult to apply BDD-based methods to hard disk data. After 2000, computers' main memory capacity grew rapidly, and in many practical cases, all the input data can be stored in the main memory. Thus many kinds of in-memory algorithms could be actively studied for data processing applications. The BDD/ZDD algorithm is a typical instance of such in-memory techniques.

In view of the above technical background, most research on BDDs/ZDDs after the winter period is not just about remaking old technologies. It also compares classical and well-known efficient methods (such as suffix trees, string matching, and frequent pattern mining) with the BDD/ZDD-based methods, in order to propose combined or improved techniques to obtain the current best performance. For example, Darwiche, who is a well-known researcher in data structures of probabilistic inference models, is very interested in the techniques of BDDs, and he recently proposed a new data structure, the SDD (Sentential Decision Diagram) [Darwiche11][39], to combine BDDs with the classical data structures of knowledge databases. As shown in this example, we had better collaborate with many researchers in various fields of information science to import the current best-known techniques into BDD/ZDD related work, and then we may develop more efficient new data structures and algorithms for discrete structure manipulation.

4. Recent Research Activity

Here we will highlight some remarkable work related to BDDs/ZDDs.

4.1 Knuth's Simpath Algorithm

As mentioned above, Knuth has presented the surprisingly fast algorithm "Simpath" [3] (Vol. 4, Fascicle 1, p. 121, or p. 254 of Vol. 4A) to construct a ZDD which represents all the paths connecting two points s and t in a given graph (not necessarily the shortest ones but ones not passing through the same point twice). This work is important because many kinds of practical problems can be efficiently solved by some variations of this algorithm. Knuth provides his own C source codes on his web page for public access, and the program is surprisingly fast. For example, in a 15×15 grid graph (420 edges in total), the number of self-avoiding paths between opposite corners is as great as 227449714676812739631826459327989863387613323440 ($\approx 2.27 \times 10^{47}$) ways. By applying the Simpath algorithm, the set of paths can be compressed into a ZDD with only

[†]"Indexing" means not only generating but also storing the data with a good structure for quick access.

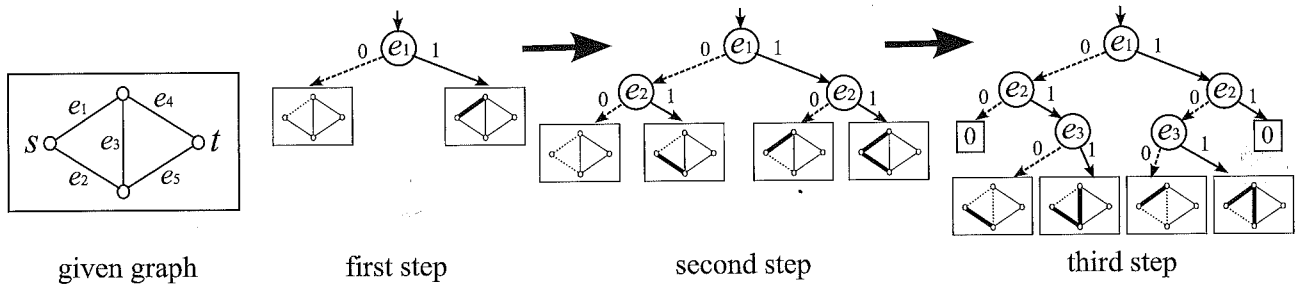


Fig. 7 Tree expansion in the Simpath algorithm.

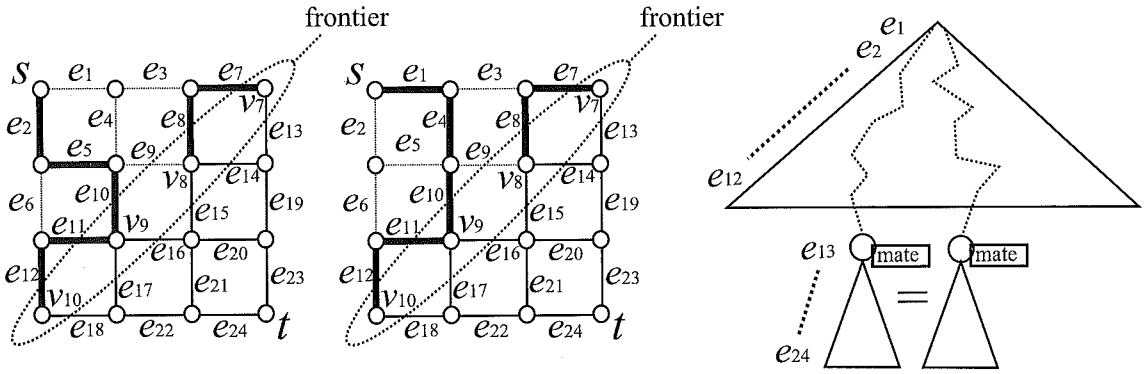


Fig. 8 Equivalent states and frontiers in the Simpath algorithm.

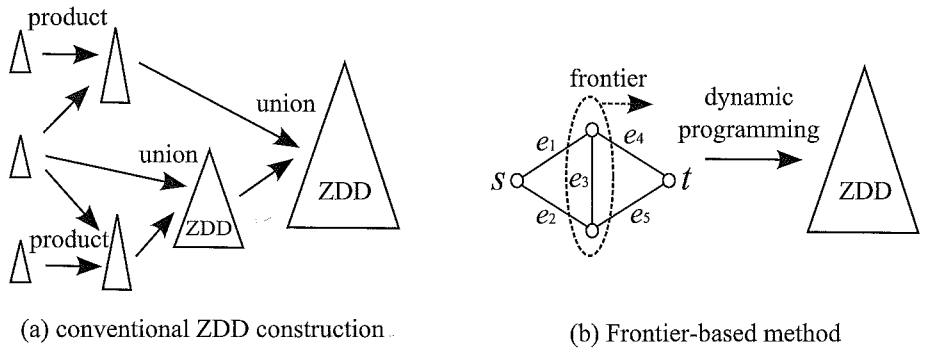
144759636 nodes, and the computation time is only a few minutes.

Figure 7 illustrates the basic mechanism of the Simpath algorithm. In the beginning, we assign a fixed ordering for all the edges, $E = \{e_1, e_2, \ldots, e_n\}$ for the given graph $G = (V, E)$. Then, we construct a binary decision tree from the top to the bottom in a breadth-first manner. In the first step, we consider two decisions 1 and 0, representing whether the edge e_1 is used in the s-t path or not. We then make two leaf nodes, each of which holds the current status of the path selection. In the second step, we visit each of the two leaf nodes and expand a new branch from the leaf, to decide whether the edge e_2 is used in the s-t path or not. Then each of the new leaves has the current status of e_1 and e_2. In this way, at the k-th level, we sequentially visit all the leaf nodes and append a $(k + 1)$-th level decision node for each case, but we may prune the branch when we find a contradiction in the current status such that the edges forming an s-t path cannot avoid generating a disjoint component or a branch. In such cases, we assign the value 0 to the leaf node and we put no more branches there. By repeating this process until the n-th level, we can construct a total decision tree including all the s-t paths. We assign the value 1 to each final leaf node which represents a solution. After that, we apply the ZDD reduction rules to each node of the decision tree from the bottom to the top to obtain a compressed ZDD.

In the above procedure, we can avoid unnecessary expansion by assigning a 0-terminal nodes to the contradicted state, but this is not enough for really fast computation. The Simpath algorithm also employs another idea of reduction to check equivalent states in the k-th level, and those equivalent nodes are merged into one node in the next expansion step. Here "equivalent states" means that the two intermediate states have completely the same requirements for the remaining undecided edges to complete the correct s-t paths. An example on a grid graph is shown in Fig. 8. Suppose that it has already been decided whether e_1 to e_{12} are used, and let us compare the two cases where $(e_2, e_5, e_7, e_8, e_{10}, e_{11}, e_{12})$ are chosen (left one) and $(e_1, e_4, e_7, e_8, e_{10}, e_{11}, e_{12})$ are chosen (right one). In both cases, we should connect the vertices v_8 and v_{10}, and connect the vertices v_7 and t by using remaining edges. Thus, the requirements of the two cases are completely the same. In the corresponding ZDD, the two sub-graphs lower than e_{13} must be identical, so we do not have to expand such sub-graphs twice if we can find such equivalent states. In many cases of the s-t path problem, a number of equivalent states appear, and the effect on the computation time is very large.

For checking the equivalence of two leaf nodes, we need only the status on a set of the vertices, specified by the dotted circle in the figure, where each vertex connects at least one decided edge and at least one undecided edge. It is enough to know which vertex is at the end point and which one is the opposite end. The Simpath algorithm prepares an array structure called "mate" to store that information at each leaf node, and registers all the mate data into a hash table for fast equivalence checking. Knuth called that set of vertices the "frontier." The frontier area moves from the start vertex to the goal vertex during computation.

The Simpath algorithm belongs to the method of dy-



(a) conventional ZDD construction (b) Frontier-based method

Fig. 9 Conventional and Frontier-based ZDD construction.

namic programming, based on the mate information on the frontier. If the frontier grows larger in the computation process, more intermediate states appear and more computation time is required. Thus, we had better keep the frontier small. The maximum size of the frontier depends on the given graph structures and the order of the edges. Empirically, planar or narrow graphs are favorable.

Recently, Iwashita et al. [35] reported that they succeeded in counting the total number of self-avoiding *s-t* paths for the 21×21 grid graph. This is the current world record and is officially registered in the On-Line Encyclopedia of Integer Sequences [40].

4.2 Frontier-Based Methods for Various Graph Problems

Knuth described in his book [3] that the Simpath algorithm can easily be modified to generate not only *s-t* paths but also Hamilton paths, directed paths, some kinds of cycles, and many other problems by slightly changing the mate data structure. We generically call such ZDD construction method "Frontier-based methods."

As illustrated in Fig. 9 (a)(b), a Frontier-based method is different from the conventional ZDD construction, which repeats primitive set operations between two ZDDs. In general, the primitive set operations are efficiently implemented based on Bryant's Apply algorithm, but do not directly use the properties of the specific problem. Frontier-based methods are sometimes much more efficient because they are a dynamic programming method based on the frontier, a kind of structural property of the given graph.

Frontier-based methods are related to mathematical work in the literature of graph theory. The *Tutte polynomial* is known as a basic invariant of a graph structure. For a given graph $G = (V, E)$, the Tutte polynomial is defined as [41]:

$$T(x, y) = \sum_{A \subseteq E} (x-1)^{\rho(E)-\rho(A)} (y-1)^{|A|-\rho(A)},$$

where $\rho(A)$ is given by subtracting the number of connecting components in a set of edges A ($\subseteq E$) from the total number of vertices $|V|$. If we could calculate the coefficients of all terms in the Tutte polynomial, it would be very powerful since various properties concerning graph connectivity can easily be evaluated. However, except for small graphs, it is too time-consuming to exactly calculate the above formula for the whole subset A.

In 1995, Sekine, Imai, et al. [42] proposed a method for generating BDDs to calculate Tutte polynomials efficiently. This method is very similar to the Simpath algorithm[†]. The differences are only that they generates BDDs instead of ZDDs, and that the "mate data" represents a set of partitions of connected components, not a pair of end points of paths. The other parts of the algorithm are similar. They also presented extensive mathematical discussions and a theoretical analysis of the complexity of the algorithm for some special classes of graph, such as planar and grid graphs [43]. Their theoretical results can also be used for the Simpath algorithm in an almost identical manner.

Unfortunately, at that time, the performance of computer hardware was much poorer than today, and BDD-related techniques were thought to be matured, so their results were mainly focused on the theoretical and mathematical viewpoints and did not consider the real computation time. Recently, Knuth proposed an efficient implementation of Simpath, so people have recognized that Frontier-based methods are interesting not only theoretically but also as state-of-the-art techniques for solving many practical problems.

Here we list graph problems which can be enumerated and indexed by a ZDD using a Frontier-based method.

- Simpath type:
  (Equivalent states are identified by end points of paths.)
  - all *s-t* paths,
  - *s-t* paths with length k,
  - k-pairs of *s-t* paths,
  - all cycles,
  - cycles with length k,
  - Hamilton paths / cycles,
  - Euler paths / cycles,
  - directed paths / cycles.
- Tutte polynomial type:
  (Equivalent states are identified by graph partitions.)

[†]Unfortunately, Knuth did not know their work when writing the Simpath algorithm for his book.

- all connected components,
- spanning trees / forests,
- Steiner trees,
- all cut sets,
- k-partitioning,
- calculating probability of connectivity

• Others:

- all cliques,
- all independent sets,
- graph colorings,
- tilings,
- perfect / imperfect matching.

These problems are strongly related to many kinds of real-life problems. For example, path enumeration is of course important in geographic information systems, and is also used for dependency analysis of a process flow chart, fault analysis of industrial systems, etc. Recently, Inoue et al. [34] discussed the design of electric power distribution systems. Such civil engineering systems are usually near to planar graphs, so the Frontier-based method is very effective in many cases. They succeeded in generating a ZDD to enumerate all the possible switching patterns in a realistic benchmark of an electric power distribution system with 468 switches. The obtained ZDD represents as many as 10^{60} of valid switching patterns but the actual ZDD size is less 100MByte, and computation time is around 30 minutes. After generating the ZDD, all valid switch patterns are compactly represented, and we can efficiently discover the switching patterns with maximum, minimum, and average cost. We can also efficiently apply additional constraints to the current solutions. In this way, Frontier-based methods can be utilized for many kinds of real-life problems.

4.3 Manipulation of 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, and sequential events. This model is also called a *language* in traditional computation theory. $\{aaa, aba, bbc, bc\}$ is an example of set of sequences obtained from 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. In 2009, Loekito, Bailey, and Pei [4] proposed *Sequence BDD (SeqBDD)*, which is a new ZDD-based data structure for representing sets of sequences. Figure 10 shows an example of a SeqBDD. This is almost same as a ZDD but differs in the variable ordering rule. SeqBDDs are based on the "lexicographical variable ordering rule," such that the variable order constraint is applied only to the 0-edge, while the 1-edge side has no ordering constraint. As shown in Fig. 11, this half-relaxed rule allows the duplication of letters in one sequence. In the SeqBDD semantics, each path in a SeqBDD corresponds to a sequence. More specifically, the top node corresponds to the head letter of the sequence, and the successive node of the 1-edge corresponds to the subsequent letters.

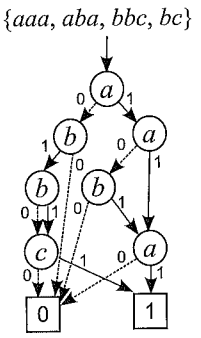


Fig. 10 A SeqBDD.

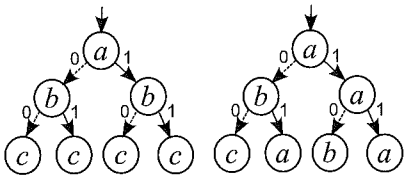


Fig. 11 Lexicographical variable ordering rule.

Table 3 Primitive SeqBDD operations.

$\emptyset$	Returns empty set. (0-terminal node)
$\{\lambda\}$	Returns the set of a null-sequence. (1-terminal node)
$P.top$	Returns the item-ID at the root node of P.
$P.onset(x)$	Selects the subset of sequences starting with letter x, and then removes x from the head of each sequence.
$P.offset(x)$	Selects the subset of sequences not starting with letter x.
$P.push(x)$	Appends x to the head of every sequence in P.
$P \cup Q$	Returns the union set.
$P \cap Q$	Returns the intersection set.
$P \setminus Q$	Returns the 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 of sequences in P and in Q)

Table 3 summarizes the primitive algebraic operations of SeqBDDs. We can see that this is very close to the ZDD algebra. The *onset, offset,* and *push* operations are slightly different from those of ordinary ZDDs. In principle, the SeqBDDs give a special meaning to the first position of a sequence. Each decision node checks the head letter of all sequences, and then divides them into two sub-sequences. Other binary operations, such as union, intersection, and difference, are almost identical. Another interesting point is that the zero-suppressed reduction rule is necessary for SeqBDD reduction. The ordinary (symmetric) BDD reduction rule conflicts with the asymmetric variable ordering rule in SeqBDDs.

A SeqBDD directly represents a set of sequences, and we do not have to know the maximum length of the sequences at the beginning. SeqBDDs are efficient especially for representing sets of sequences containing both very long

and short sequences. Denzumi et al. [44] presented the theoretical relationship between SeqBDDs and acyclic automata in terms of computation time and space. They also presented a technique "Suffix-DD" [45], which is a suffix-tree manipulation based on SeqBDDs. This method efficiently constructs an index of all substrings for a given long text string. The techniques of automata and suffix-trees are commonly used in string data applications, so we hope that many of those applications will be accelerated by using SeqBDDs.

4.4 Manipulation of Sets of Permutations

Now we focus on manipulating permutations, as one of the most important discrete structure models. Permutations and combinations are two basic concepts in elementary combinatorics and discrete mathematics [46]. Permutations appear in various problems such as sorting, ordering, matching, coding and many other real-life situations. Recently, the author has developed a new type of decision diagram, named "πDD" [5], for the compact and canonical representation of a set of permutations. Figure 12 shows an example of a multi-rooted πDD corresponding to several sets of permutations. The πDD method is the first practical idea for the efficient manipulation of sets of permutations on the basis of decision diagrams. This data structure can compress a large number of permutations into a compact and canonical representation.

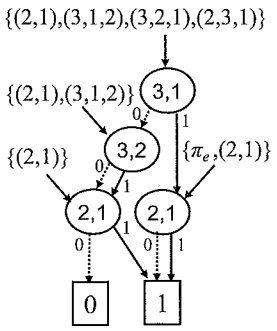


Fig. 12 Example of a shared πDD.

Table 4 Primitive πDD operations.

$\emptyset$	Returns empty set. (0-terminal node)
$\{\pi_e\}$	Returns the set of an identity permutation. (1-terminal node)
P.top	Returns IDs (x, y) at the root node of P.
$P \cup Q$	Returns $\{\pi \mid \pi \in P \text{ or } \pi \in Q\}$.
$P \cap Q$	Returns $\{\pi \mid \pi \in P,\ \pi \in Q\}$.
$P \setminus Q$	Returns $\{\pi \mid \pi \in P,\ \pi \notin Q\}$.
$P.\tau(x, y)$	Returns $P \cdot \tau_{(x,y)}$.
$P * Q$	Returns $\{\alpha\beta \mid \alpha \in P,\ \beta \in Q\}$.
$P.cofact(x, y)$	Returns $\{\pi\tau_{(x,y)} \mid \pi \in P,\ x\pi = y\}$.
P.count	Counts the number of permutations.

Similar to ordinary BDDs and ZDDs, πDDs have efficient algebraic set operations, such as union, intersection, and difference. Table 4 summarizes the primitive operations used in πDDs for manipulating sets of permutations. There is also a special Cartesian product operation which generates all possible composite permutations (cascades of two permutations) for two given sets of permutations. This operation is beautiful and is a powerful method for solving various problems in permutation space. For example, we can represent the primitive moves of Rubik's Cube with a small πDD, and by simply multiplying this πDD by itself k times, we can generate a single canonical πDD representing all possible positions reachable within k moves. The computation time depends on the size of the πDD, which is sometimes much smaller than the number of positions. Figure 13 shows an example of a product operation for two πDDs whose items are disjoint. In this case, even though the number of permutations increases multiplicatively, the size of the πDD increases only additively. Since the computation time also depends on the size of the πDD, in such cases the effectiveness of the πDD-based method increases exponentially when compared with using an explicit data structure. Once we have generated πDDs for a problem, we can easily apply various analysis or testing techniques, such as counting the exact number of permutations, exploring satisfiable permutations for a given constraint and calculating the minimal or average cost of all permutations.

Recently, Kawahara et al. [47] applied πDDs and their algebraic operations to the analysis of primitive sorting networks of width n. They succeeded in calculating that the

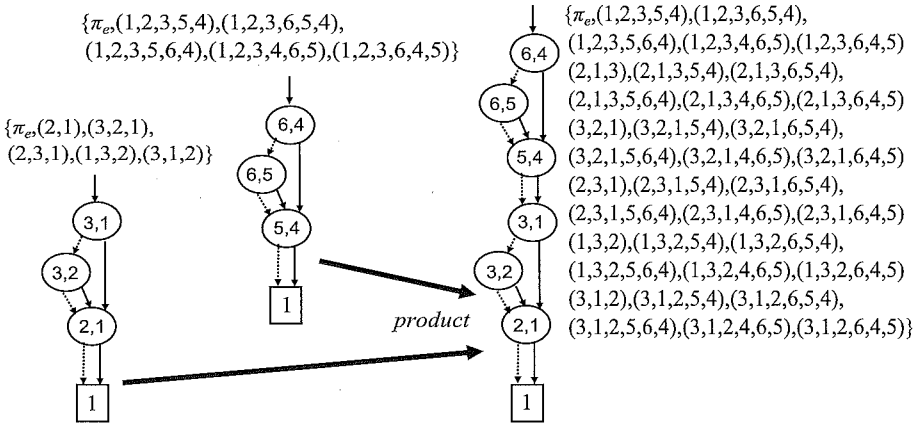


Fig. 13 Example of the Cartesian product operation for πDDs.

number of ways to construct minimum primitive sorting networks for $n = 13$ is 2752596959306389652, which was not previously known. This result is officially registered in the On-Line Encyclopedia of Integer Sequences [40].

Soeken et al. [48] reported that πDDs are also useful for reversible logic design. A reversible Boolean function is a special type of Boolean function which has the same numbers of inputs and outputs with a bijective relation between them. Since a bijection corresponds to a permutation, a set of reversible functions can be represented by a πDD. Reversible logic is strongly related to loss-less encoding and quantum computation [49], so there will be some interesting future work.

5. Concluding Remarks

We have given a brief history and described recent research on the techniques of BDDs and ZDDs. As shown in the above sections, there are very many interesting and useful ideas about decision diagrams and discrete structure manipulation. The author has proposed a new research project which focuses on these techniques. The proposal was accepted in 2009 as an ERATO project, a nation-wide project supported by the Japanese scientific research agency, and it is now ongoing [50]. In this project, we consider BDDs/ZDDs not just as data structures, but regard them as integrated manipulation systems for many types of discrete structures. Discrete structure manipulation is a fundamental task in various applications of computers. Therefore, its acceleration will have a great effect on society.

Acknowledgements

The author would like to express his gratitude to all the members and collaborative researchers of the JST ERATO Minato Discrete Structure Manipulation System Project for their fruitful discussions. The author also would like to thank the chair and the committee members of the Technical Group on Computation (COMP) of IEICE, for providing the opportunity to write this survey paper.

References

[1] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, 1986.
[2] S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," Proc. 30th ACM/IEEE Design Automation Conference (DAC'93), pp.272–277, 1993.
[3] D.E. Knuth, The Art of Computer Programming: Bitwise Tricks & Techniques; Binary Decision Diagrams, Addison-Wesley, 2009.
[4] E. Loekito, J. Bailey, and J. Pei, "A binary decision diagram based approach for mining frequent subsequences," Knowledge and Information Systems, vol.24, no.2, pp.235–268, 2010.
[5] S. Minato, "πDD: A new decision diagram for efficient problem solving in permutation space," in Theory and Applications of Satisfiability Testing - SAT 2011, ed. K. Sakallah and L. Simon, Lecture Notes in Computer Science, vol.6695, pp.90–104, Springer Berlin Heidelberg, 2011.
[6] S.B. Akers, "Binary decision diagrams," IEEE Trans. Comput., vol.C-27, no.6, pp.509–516, 1978.
[7] R. Yoshinaka, J. Kawahara, S. Denzumi, H. Arimura, and S. Minato, "Counterexamples to the long-standing conjecture on the complexity of bdd binary operations," Inf. Process. Lett., vol.112, no.16, pp.636–640, 2012.
[8] S. Minato, "Zero-suppressed BDDs and their applications," International J. Software Tools for Technology Transfer, vol.3, pp.156–170, 2001.
[9] O. Coudert, "Solving graph optimization problems with ZBDDs," Proc. ACM/IEEE European Design and Test Conference (ED&TC '97), pp.224–228, 1997.
[10] E. Loekit and J. Bailey, "Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams," Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2006), pp.307–316, 2006.
[11] S. Minato, K. Satoh, and T. Sato, "Compiling bayesian networks by symbolic probability calculation based on zero-suppressed BDDs," Proc. 20th International Joint Conference of Artificial Intelligence (IJCAI-2007), pp.2550–2555, 2007.
[12] M. Ishihata, Y. Kameya, T. Sato, and S. Minato, "Propositionalizing the EM algorithm by BDDs," Proc. 18th International Conference on Inductive Logic Programming (ILP 2008), Sept. 2008.
[13] K.S. Brace, R.L. Rudell, and R.E. Bryant, "Efficient implementation of a BDD package," Proc. 27th ACM/IEEE Design Automation Conference, DAC '90, pp.40–45, New York, NY, USA, 1990.
[14] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagrm with attributed edges for efficient Boolean function manipulation," Proc. 27th ACM/IEEE Design Automation Conference, pp.52–57, 1990.
[15] M. Fujita, Y. Matsunaga, and T. Kakuda, "On variable ordering of binary decision diagrams for the application of multi-level logic synthesis," Proc. Conference on European Design Automation, EURO-DAC '91, Los Alamitos, CA, USA, pp.50–54, 1991.
[16] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," Proc. 1993 IEEE International Conference on Computer-Aided Design, ICCAD '93, pp.42–47, Los Alamitos, CA, USA, 1993.
[17] S. Tani, K. Hamaguchi, and S. Yajima, "The complexity of the optimal variable ordering problems of shared binary decision diagrams," in Algorithms and Computation, ed. K. Ng, P. Raghavan, N. Balasubramanian, and F. Chin, Lect. Notes Comput. Sci., vol.762, pp.389–398, 1993.
[18] R. Drechsler, N. Drechsler, and W. Günther, "Fast exact minimization of BDDs," Proc. 35th Design Automation Conference, DAC '98, pp.200–205, New York, NY, USA, 1998.
[19] F. Somenzi, "Cudd: Cu decision diagram package," 1997. http://vlsi.colorado.edu/~fabio/CUDD/
[20] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and implementation of boolean comparison method based on binary decision diagrams," Proc. IEEE International Conf. on Computer-Aided Design (ICCAD-88), pp.2–5, 1988.
[21] S. Malik, A. Wang, R.K. Brayton, and A. S.-Vincentelli, "Logic verification using binary decision diagrams in a logic synthesis environment," Proc. IEEE International Conf. on Computer-Aided Design (ICCAD-88), pp.6–9, 1988.
[22] O. Coudert and J.C. Madre, "A unified framework for the formal verification of sequential circuits," Proc. IEEE International Conf. on Computer-Aided Design (ICCAD-90), pp.126–129, 1990.
[23] Y. Matsunaga and M. Fujita, "Multi-level logic optimization using binary decision diagrams," Proc. IEEE International Conf. on Computer-Aided Design (ICCAD-89), pp.556–559, 1989.
[24] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill, "Sequential circuit verification using symbolic model checking," Proc. 27th ACM/IEEE Design Automation Conference, DAC '90, New York, NY, USA, pp.46–51, ACM, 1990.
[25] K.L. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.
[26] J.P.M. Silva and K.A. Sakallah, "Grasp-a new search algorithm for

satisfiability," Proc. IEEE/ACM International Conf. on Computer-Aided Design (ICCAD-96), pp.220–227, 1996.
[27] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using sat procedures instead of BDDs," Proc. 36th ACM/IEEE Design Automation Conference, DAC '99, pp.317–320, New York, NY, USA, 1999.
[28] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient sat solver," Proc. 38th Annual Design Automation Conference, DAC '01, pp.530–535, New York, NY, USA, 2001.
[29] H. Okuno, S. Minato, and H. Isozaki, "On the properties of combination set operations," Inf. Procss. Lett., vol.66, pp.195–199, 1998.
[30] S. Minato, "VSOP (valued-sum-of-products) calculator for knowledge processing based on zero-suppressed BDDs," in Federation over the Web, ed. K. Jantke, A. Lunzer, N. Spyratos, and Y. Tanaka, Lect. Notes Comput. Sci., vol.3847, pp.40–58, 2006.
[31] 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 Advances in Knowledge Discovery and Data Mining, ed. T. Washio, E. Suzuki, K. Ting, and A. Inokuchi, Lect. Notes Comput. Sci., vol.5012, pp.234–246, 2008.
[32] Y. Sakurai, S. Ueda, A. Iwasaki, S. Minato, and M. Yokoo, "A compact representation scheme of coalitional games based on multi-terminal zero-suppressed binary decision diagrams," in Agents in Principle, Agents in Practice, ed. D. Kinny, J.J. Hsu, G. Governatori, and A. Ghose, Lect. Notes Comput. Sci., vol.7047, pp.4–18, 2011.
[33] R. Yoshinaka, T. Saitoh, J. Kawahara, K. Tsuruma, H. Iwashita, and S. Minato, "Finding all solutions and instances of numberlink and slitherlink by ZDDs," Algorithms, vol.5, no.2, pp.176–213, 2012.
[34] T. Inoue, K. Takano, T. Watanabe, J. Kawahara, R. Yoshinaka, A. Kishimoto, K. Tsuda, S. Minato, and Y. Hayashi, "Loss minimization of power distribution networks with guaranteed error," Hokkaido University, Division of Computer Science, TCS Technical Reports, vol.TCS-TR-A-10-59, 2012.
[35] H. Iwashita, J. Kawahara, and S. Minato, "ZDD-based computation of the number of paths in a graph," Hokkaido University, Division of Computer Science, TCS Technical Reports, vol.TCS-TR-A-10-60, 2012.
[36] J.C. Madre and O. Coudert, "A logically complete reasoning maintenance system based on a logical constraint solver," Proc. 12th international joint conference on Artificial intelligence - vol.1, San Francisco, CA, USA, pp.294–299, Morgan Kaufmann Publishers Inc., 1991.
[37] O. Coudert and J.C. Madre, "Towards an interactive fault tree analyser," Proc. IASTED International Conference on Reliability, Quality Control and Risk Assessment, 1992.
[38] O. Coudert and J.C. Madre, "Implicit and incremental computation of primes and essential primes of boolean functions," Proc. 29th ACM/IEEE Design Automation Conference, DAC '92, pp.36–39, Los Alamitos, CA, USA, 1992.
[39] A. Darwiche, "SDD: A new canonical representation of propositional knowledge bases," Proc. 22nd International Joint Conference of Artificial Intelligence (IJCAI-2011), pp.819–826, 2011.
[40] "The on-line encyclopedia of integer sequences," 2011. https://oeis.org/
[41] D.J.A. Welsh, "Complexity: Knots, colourings and counting," London Mathematical Society Lecture Note Series, vol.186, pp.372–390, 1993.
[42] K. Sekine, H. Imai, and S. Tani, "Computing the tutte polynomial of a graph of moderate size," in Algorithms and Computations, ed. J. Staples, P. Eades, N. Katoh, and A. Moffat, Lect. Notes Comput. Sci., vol.1004, pp.224–233, 1995.
[43] H. Imai, S. Iwata, K. Sekine, and K. Yoshida, "Combinatorial and geometric approaches to counting problems on linear matroids, graphic arrangements, and partial orders," Computing and Combinatorics, ed. J.Y. Cai and C. Wong, Lect. Notes Comput. Sci., vol.1090, pp.68–80, 1996.
[44] S. Denzumi, R. Yoshinaka, H. Arimura, and S. Minato, "Notes on sequence binary decision diagrams: Relationship to acyclic automata and complexities of binary set operations," Proc. Prague Stringology Conference 2011, ed. J. Holub and J. Žďárek, Czech Technical University in Prague, Czech Republic, pp.147–161, 2011.
[45] S. Denzumi, H. Arimura, and S. Minato, "Substring indices based on sequence bdds," Hokkaido University, Division of Computer Science, TCS Technical Reports, vol.TCS-TR-A-10-42, 2010.
[46] D.E. Knuth, "Combinatorial properties of permutations," The Art of Computer Programming, ch. 5.1, pp.11–72, Addison-Wesley, 1998.
[47] J. Kawahara, T. Saitoh, R. Yoshinaka, and S. Minato, "Counting primitive sorting networks by pidds," Hokkaido University, Division of Computer Science, TCS Technical Reports, vol.TCS-TR-A-11-54, 2011.
[48] M. Soeken, R. Wille, S. Minato, and R. Drechsler, "Using πdds in the design of reversible circuits," Preliminary Proc. 4th Workshop on Reversible Computation (RC-2012), pp.205–211, 2012.
[49] S. Yamashita, S. Minato, and D.M. Miller, "DDMF: An efficient decision diagram structure for design verification of quantum circuits under a practical restriction," IEICE Trans. Fundamentals, vol.E91-A, no.12, pp.3793–3802, Dec. 2008.
[50] S. Minato, "Overview of ERATO Minato project: The art of discrete structure manipulation between science and engineering," New Generation Computing, vol.29, no.2, pp.223–228, 2011.

Shin-ichi Minato is a Professor at the Graduate School of Information Science and Technology, Hokkaido University. He also serves as the Research Director of the ERATO MINATO Discrete Structure Manipulation System Project, executed by JST. He received the B.E., M.E., and D.E. degrees in Information Science from Kyoto University in 1988, 1990, and 1995, respectively. He worked for NTT Laboratories from 1990 until 2004. He was a Visiting Scholar at the Computer Science Department of Stanford University in 1997. He joined Hokkaido University as an Associate Professor in 2004, and has been a Professor since October 2010. He has published "Binary Decision Diagrams and Applications for VLSI CAD" (Kluwer, 1995). He is a member of IEEE, IPSJ, and JSAI.