Thesis submitted to obtain the title
Doctor of Information Science
平成30年度　博士学位論文

Implicit Subset Enumeration by Top-Down Construction of Decision Diagrams and Its Applications to Combinatorial Problems

決定グラフのトップダウン構築による暗黙的部分集合列挙とその組合せ問題への応用

Hirofumi Suzuki

鈴木 浩史

Large-Scale Knowledge Processing Laboratory,
Division of Computer Science and Information Technology,
Graduate School of Information Science and Technology
Hokkaido University

北海道大学 大学院情報科学研究科
情報理工学専攻 大規模知識処理研究室

February 2019

Abstract

Real-life involves various selections of objects, i.e., *combinations* which make *subsets* of selected objects. As a popular example, a solution of a *puzzle* is a subset of components such as numbers, lines, and positions. Road networks and chemical compounds are often represented by a subset of connections, i.e., *graphs*. Especially, life consists of itemsets such as foods, tools, and clothes. By the concept of the combination, *combinatorial problems* model for a broad range of issues. A combinatorial problem requires obtaining desirable subsets or objective values with given *constraints* specifying possible combinations. For example, the *knapsack problem* (KP) requires obtaining a subset of items with the maximum value within a capacity.

For a long time, many researchers have tried to efficiently solve various types of combinatorial problems. *Generation* is to generate subsets satisfying specified constraints, such as solutions of puzzles and investment plans within a budget. *Evaluation* is to evaluate various properties of interesting subsets, which gives us knowledge such as number of chemical compounds, reliability of network systems, and so on. *Optimization* is to obtain subsets that are optimal in given criteria such as itemsets with maximum value and network systems with minimum cost.

On the other hand, the *enumeration* is known as a special case of the generation. It requires generating all subsets satisfying given constraints without omission and duplication. Especially, the enumeration is involved in various combinatorial problems, for example, the computation of the Tutte polynomial by using all the spanning trees in a given graph. In addition, the enumeration gives us various useful functions such as random sampling, counting, and optimizing. However, we have a concerned issue called the *combinatorial explosion* which means the rapid growth of the number of possible combinations. Due to the combinatorial explosion, the enumeration can take exponential time: If we have n objects, the number of possible combinations is 2^n.

Therefore, we deal with the enumeration avoiding the *explicit enumeration*. It outputs the solutions one by one, which easily causes the combinatorial explosion. In particular, we aim to conduct the *implicit enumeration* that generates subsets by using a compact form. It has a potential to efficiently enumerate huge number of subsets and quickly perform various useful functions.

For the implicit enumeration, we focus on the *binary decision diagram* (BDD) and the *zero-suppressed binary decision diagram* (ZDD). The BDD is a data structure for representing *Boolean functions* compactly. Because a Boolean function is an indicator function of a *set family*, BDD construction implies the implicit enumeration. The ZDD is a derivation of the BDD to represent set families compactly. Especially, on representing sparse set families, ZDDs are often much smaller than BDDs empirically. Moreover, the BDD and the ZDD has various useful functions such as set operations, counting the number of subsets, calculating the occurrence probability, and obtaining an optimal subset with a linear criterion. Therefore, they have been applied to the various combinatorial problems, for example, the *maximum independent set*, N-queens problem, evacuation planning, and so on.

An important issue for the implicit subset enumeration by BDDs and ZDDs is how to construct BDDs and ZDDs. The *frontier-based search* (FBS) is a general procedure for constructing BDDs and ZDDs representing constrained subgraphs on a given graph. A characteristic thing of the FBS is to conduct a top-down construction of BDDs and ZDDs. In addition, there is a generic library called TdZdd for the FBS implementation. However, there are no formulation of a top-down construction of BDDs and ZDDs.

In this thesis, we formulate a general framework for a top-down construction of BDDs and ZDDs. The framework conducts the implicit enumeration of various constrained subsets, whereas the FBS can deal with only graph structures. We named the framework the TD-DD. In addition, we discuss the time complexity of the TD-DD. Subsequently, we also apply the TD-DD for combinatorial problems related to *Minesweeper* puzzles, *strongly connected spanning subgraphs*, and *Pareto-optimal* itemsets with a knapsack constraint.

In Chapter 4, we deal with the enumeration related to the Minesweeper puzzle. The Minesweeper puzzle is a popular puzzle game to open all safety cells avoiding mine cells while referring hints on a given board. The previous studies on the Minesweeper puzzle are interested in the existence or the number of feasible mine assignments on a fixed board. However, there are no research problems to generate specific solutions. Therefore, we define the *Minesweeper generation problem* (MGP) that requires generating all the feasible mine assignments of a given fixed board. Subsequently, we propose two algorithms to solve the MGP using the family algebra of ZDDs or the TD-DD. We also apply them some related problems and computation of mine probability. The experimental results showed efficient performances of the algorithm using TD-DD.

In Chapter 5, we deal with the strongly connected spanning subgraphs (SCSSs) on directed graphs: if every vertices are reachable each other on a subgraph, the subgraph is an SCSS of an original graph. SCSSs have often been applied to issues on network systems: for example, computation of *strongly connected reliability* (SCR) and obtaining the minimum SCSS (min-SCSS) for an efficient visualization. Therefore, we

propose an algorithm to implicit enumerate SCSSs by the TD-DD. We also apply the algorithm to the computation of the SCR and obtaining the min-SCSS. The experimental results showed that our algorithm practically ran on real-world graphs with a few hundred edges. Moreover, we succeeded to compute the exact SCR for various graphs, which was previously impossible.

In Chapter 6, we deal with the *0-1 multi-objective knapsack problem* (MKP) which is a *multi-objective combinatorial optimization problem* (MOCO). The MKP is a derivation of the KP with multiple criteria. The MOCO causes multiple optimal subsets with different values each other, which are said to be Pareto-optimal. Therefore, we propose an algorithm using the TD-DD of ZDDs to implicitly enumerate all Pareto-optimal solutions of the MKP. The algorithm utilizes the TD-DD in the preprocess to enumerate all feasible solutions. Especially, we use an extended framework of the TD-DD derived from an existing algorithm with novel pruning techniques. In addition, we propose new pruning techniques based on properties of the ZDD. The experimental results showed that our algorithm is faster than the existing algorithm on various instances.

In Chapter 7, we summarize the results in this thesis and shows future works.

Contents

Chapter 1

Introduction

1.1 Background

A *combination* is a selection of non-duplicated objects from a collection of distinct objects. On a set of objects, the set of selected objects by a combination is called a *subset*. The concept of the combination and the subset has been inseparable from the real-world. The followings are typical examples:

- *Puzzle*: Many puzzles ask us to find a subset of components such as numbers, lines, positions, and pairs of them. Sudoku, number links, and N-queens problem are just such puzzles.
- *Graph*: A graph represents connections of things by using combinatorial structures *vertices* and *edges*. Road networks, chemical compounds, and other various structures can often be represented by graphs.
- *Itemset*: The real-life has consisted in selections from various itemsets such as foods, tools, and clothes. On the real situation, itemsets are involved in purchase data, selecting investments, and so on.

By the concept of the combination and the subset, *combinatorial problems* can model for a broad range of issues in the real-world. A combinatorial problem requires obtaining desirable subsets or objective values with given *constraints* that specify possible combinations. For example, the *knapsack problem* (KP) is one of the famous combinatorial problems. Given a capacity and a set of items having a value and a weight, the KP requires obtaining a subset of the items with the maximum total value within the capacity. The KP appears in various situations such as selecting investments, resource management in software, and the rest.

Because of applying combinatorial problems to real-world issues, many researchers have tried to efficiently solve various problems for a long time. For example, the following types of problems have been widely studied.

- *Generation*: Generate subsets satisfying given constraints: for example, solutions of puzzles such as N-queens problem [14]. This type of problems produces various fundamental approaches to combinatorial problems such as searching, branching, and pruning.

- *Evaluation*: Evaluate the total number, the occurrence probability, and other properties of subsets satisfying given constraints. Polya's theory [52] is a famous result on counting chemical compounds. On analyzing networks, computation of the Tutte polynomial [60] is useful to the reliability evaluation [54].

- *Optimization*: Obtain subsets that are said to be optimal in given criteria. For example, the KP is a classically studied optimization problem. If the number of criteria is two or more, the problem is said to be a *multi-objective combinatorial optimization problem* (MOCO) [58].

On the other hand, the *enumeration* is known as a special case of the generation. The enumeration requires generating all subsets satisfying given constraints without omission and duplication. Moreover, the enumeration is involved in various combinatorial problems. For example, the Tutte polynomial can be calculated by the enumeration of all the spanning trees in a given graph. The MOCO takes multiple optimal subsets that have different values each other. Namely, the MOCO requires the enumeration of *Pareto-optimal* solutions. In addition, we can apply the enumeration to other applications such as random sampling [38], finding frequent itemsets [62], counting, and optimizing. However, there is a most concerned issue on the enumeration called the *combinatorial explosion*. It means the rapid growth of the number of possible combinations. Due to the combinatorial explosion, the enumeration can take an enormous amount of time: If there are n objects, the number of their possible combinations is 2^n which is exponential. Even if subsets must have exactly k objects, the number of possible combinations is

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1}$$

that is still huge depending on k.

In this thesis, we deal with the enumeration avoiding the *explicit enumeration* such as outputting the solutions one by one, which easily causes the combinatorial explosion. Therefore, we aim to conduct the *implicit enumeration* that generates subsets by using a compact form, for example, a compressed data structure. The implicit enumeration has a potential to efficiently enumerate huge number of subsets and quickly perform useful functions for subsets such as random sampling, counting, optimizing, and so on.

1.2 Related Work

The *backtracking* is a classic approach to solve various combinatorial problems related to the generation and the optimization. The backtracking is a branching algorithm which enumerates partial subsets explicitly like the exhaustive search by using strategic branching and pruning techniques. For a long time, representative applications of the backtracking are puzzles such as N-queens problem. Although the backtracking tends to be faster than the brute force approach, it is not always true that the approach avoids the combinatorial explosion.

The *reverse search* [5] is a polite enumeration technique using recursive search without pruning. For each phase of the reverse search, it outputs a subset that is different from the previous ones, i.e., wasted searches do not occur during the explicit enumeration. However, possible subjects of the reverse search are still widely being studied. The reverse search has often been applied to the field of the data mining, for example, community discovering via pseudo clique enumeration [61], mining of frequent patterns from graph sequences [25], and the rest.

On the other hand, the *binary decision diagram* (BDD) [15] achieves the implicit subset enumeration. The BDD is a data structure for representing *Boolean functions* compactly. Because a Boolean function is an indicator function of a *set family*, i.e., a set of subsets, the construction of BDDs corresponds to the implicit subset enumeration. In addition, the *zero-suppressed binary decision diagram* (ZDD) [47] is a derivation of the BDD to represent set families compactly. Especially, on representing sparse set families, ZDDs are often much smaller than BDDs empirically.

The BDD and the ZDD have various useful functions such as set operations, counting the total number of subsets, calculating the occurrence probability, and obtaining an optimal subset with a linear criterion. These functions are efficiently performed on compressed forms. Therefore, the BDD and the ZDD have been applied to various combinatorial problems for example the *maximum independent set problem* [10], the computation of Tutte polynomial [54], the computation of a *network reliability* [24], N-queens problem [47], the *maximum clique problem* [17], and the rest.

Whereas the mainstream of constructing BDDs and ZDDs was multi-step by using set operations in combination, the algorithm in the literature [54] was a one-step algorithm by a top-down construction of BDDs. In addition, the literature [40] shows an algorithm for enumerating all s-t paths in a given graph by a top-down construction of ZDDs. These algorithms are generalized as the *frontier-based search* (FBS) [34] that constructs BDDs and ZDDs for the implicit subgraph enumeration. The FBS has been widely used for solving real-world issues such as loss minimization of power distribution networks [26] and region partition within a disparity bound [33].

A top-down construction of BDDs and ZDDs has also been developed for solving the

knapsack problem, maximum independent set problem, and the rest [9]. Moreover, a generic library TdZdd [29] has been built for the implementation of a top-down construction of BDDs and ZDDs. However, there are no research to formulate a top-down construction of BDDs and ZDDs. This is an important issue for the implicit subset enumeration.

1.3 Our Contributions

In this thesis, we have key contributions to the implicit subset enumeration as follows: We formulate the general framework TD-DD for a top-down construction of BDDs and ZDDs. The TD-DD conducts the implicit enumeration of various constrained subsets. We also discuss about the time complexity of the TD-DD. Subsequently, using the TD-DD, we uniformly deal with the following three types of combinatorial structures:

1. Mine assignments of *Minesweeper* puzzles with fixed hints
2. *Strongly connected spanning subgraphs* on directed graphs
3. *Pareto-optimal* itemsets that satisfy a knapsack constraint

The contributions for each combinatorial structure above are summarized as follows:

1. Implicit enumeration of all the feasible mine assignments of Minesweeper puzzles with fixed hints: The Minesweeper puzzle is a popular puzzle game. Its goal is to open all safety cells avoiding mine cells while referring hints on a given grid board. The previous studies on the Minesweeper puzzle are interested in the existence or the number of feasible mine assignments on a board with fixed hints. On the other hand, there are no research problems related to the generation of specific mine assignments. Therefore, we define the *Minesweeper generation problem* (MGP) that requires generating all the feasible mine assignments of a given board with fixed hints. Especially, we consider the generalized board using graph structures. We propose two approaches using ZDDs to solve the MGP. One is a naive combinatorial approach combined with the family algebra. The other is a degree constraint approach using one-path algorithm by the TD-DD. We also show its applications to the previously studied problems and a strategy for playing Minesweeper puzzles. The experimental results showed that the approach using the TD-DD is better for various instances [1].

2. Implicit enumeration of the strongly connected spanning subgraphs (SCSSs) on directed graphs: On a directed graph, if every vertices are reachable each other,

[1] This result has been published in [55]

the graph is said to be strongly connected and spanning. On computer networks, it means that every computer can communicate each other. Given a directed graph, an SCSS of the graph is a subgraph that maintains strongly connected and spanning. SCSSs have often been applied to various issues on network systems: for example, computation of *strongly connected reliability* (SCR) [13] for analyzing network systems, obtaining the minimum SCSS (min-SCSS) [69] for an efficient visualization, and the rest. Therefore, we propose an algorithm to implicit enumerate SCSSs by the TD-DD of BDD for solving the issues above. Once a BDD for SCSSs is obtained, we can easily compute exact SCR and obtain the min-SCSS by functions of the BDD. The experimental results showed that our algorithm ran on real-world graphs with a few hundred edges. Moreover, we succeeded to compute the exact SCR for various graphs, which was previously impossible [2].

3. Implicit enumeration of the Pareto-optimal solutions of the *0-1 multi-objective knapsack problem* (MKP): The MKP is a derivation of the KP with multiple criteria and is central to the MOCO. For solving the MKP, an efficient dynamic programming (DP) algorithm with novel pruning techniques has been proposed in the literature [7]. In contrast, we propose an algorithm based on the TD-DD for constructing a ZDD representing all the Pareto-optimal solutions. First, in the preprocess, the algorithm uses the TD-DD for constructing a ZDD representing all the feasible solutions. Subsequently, the algorithm uses an extended framework of the TD-DD derived from the existing DP algorithm. An important idea of the algorithm is to use new pruning techniques based on properties of the preprocessed ZDD, which have better performance than existing ones. In addition, we propose a new heuristic of item reordering which accelerates our algorithm. The experimental results showed that our algorithm is faster than the existing DP algorithm on various types of instances [3].

1.4 Thesis Organization

In the following Chapter 2, we introduce definitions and notations about sets, set families, graphs, and decision diagrams, which are our main tools used in this thesis. Especially, we introduce various useful functions of decision diagrams. Chapter 3 provides a framework, named TD-DD, of the top-down construction of decision diagrams. In Chapter 4, we apply the framework to solve combinatorial problems related to the

[2] This result has been published in [56]
[3] This result has been published in [57]

Minesweeper puzzle. In Chapter 5, we deal with SCSSs and its applications by TDDD. In Chapter 6, we propose a new framework of top-down construction of decision diagrams for the 0-1 multi-objective knapsack problem. Chapter 7 summarizes the results in this thesis and shows future works.

Chapter 2

Preliminaries

In this chapter, we introduce required definitions and notations to describe the contributions in this thesis. More precisely, we first introduce sets and set families that are the main concepts of this thesis. Next, we introduce graphs because they appear in our contributions frequently. Finally, we introduce decision diagrams that are our main tools to solve combinatorial problems efficiently.

2.1 Sets and Set Families

We first introduce basic definitions and notations of sets and set families.

Definition 2.1.1. *A* set *is a collection of distinct objects. Each object included in a set is called an* element *of the set, and can be anything: for example, numbers, letters, and other sets. Given a set S, if S has two elements denoted by a and b, we denote* $S = \{a, b\}$ *or* $S = \{b, a\}$*, i.e., the order of the elements is arbitrary. If x is an element of S, we denote* $x \in S$*, otherwise* $x \notin S$*. The number of elements in S is called the* cardinality *of S, which is denoted by* $|S|$*. If* $|S| = 0$*, S is said to be the* empty set*, which is denoted by the special symbol* $\emptyset$*.*

Example 2.1.1. *The natural numbers at most* 5 *make a set* $S = \{1, 2, 3, 4, 5\}$ *whose cardinality is* $|S| = 5$*. For an integer x,* $x \in S$ *if* $1 \leq x \leq 5$*, otherwise* $x \notin S$*.*

Definition 2.1.2. *Given two sets A and B, if* $x \in A$ *for any element* $x \in B$*, B is called a* subset *of A. If B is a subset of A, we denote* $B \subseteq A$*.*

Example 2.1.2. *For a set* $A = \{a, b, c\}$*,* $\{a, b\} \subseteq A$*,* $\{c\} \subseteq A$*, and* $\emptyset \subseteq A$*.*

Definition 2.1.3. *A* family *is a set each of whose element is another set.*

Example 2.1.3. *A set* $\mathcal{S} = \{\{a, b\}, \{a, c\}, \{a, b, c\}\}$ *is a set family.*

Definition 2.1.4. *Given a set S, the* power set *of S is a set family that has any subsets of S as an element, which is denoted by 2^S.*

Example 2.1.4. *The power set of $S = \{a,b\}$ is $2^S = \{\emptyset, \{a\}, \{b\}, \{a,b\}\}$.*

Definition 2.1.5. *Given two sets A and B, the* union *operation is defined as $A \cup B := \{x \mid x \in A \text{ or } x \in B\}$, the* intersection *operation is defined as $A \cap B := \{x \mid x \in A \text{ and } x \in B\}$, and the* difference *operation is defined as $A \setminus B := \{x \mid x \in A, x \notin B\}$.*

Example 2.1.5. *Let $A = \{a,b,c\}$ and $B = \{b,d\}$. Then $A \cup B = \{a,b,c,d\}$, $A \cap B = \{b\}$, and $A \setminus B = \{a,c\}$.*

Definition 2.1.6. *$\mathbb{Z}$ is a set of all integers. $\mathbb{Z}_{\geq 0}$ is a set of integers which are greater than or equal to zero, namely, $\mathbb{Z}_{\geq 0} := \{x \in \mathbb{Z} \mid x \geq 0\}$. $\mathbb{N}$ is a set of positive integers, namely, $\mathbb{N} := \{x \in \mathbb{Z} \mid x > 0\}$.*

Hereinafter, let $[n] := \{1, \ldots, n\}$ for an integer $n \in \mathbb{N}$. For convenience, $[n] = \emptyset$ if $n \leq 0$.

2.2 Graphs

We also introduce basic definitions and notations of graphs. In the following, given a set S, we denote $[S]^2 := \{\{x,y\} \mid x,y \in S, x \neq y\}$ and $\langle S \rangle^2 := \{(x,y) \mid x,y \in S, x \neq y\}$.

Definition 2.2.1. *A* graph *(*simple graph *strictly) is an ordered pair $G = (V,E)$ with a set of* vertices *V and a set of* edges *E where $E \subseteq [V]^2$ or $E \subseteq \langle V \rangle^2$. If $E \subseteq [V]^2$, G is said to be* undirected*, otherwise* directed*. For each edge $\{u,v\} \in E$ or $(u,v) \in E$, u and v are the* endpoint *of the edge.*

Definition 2.2.2. *For any pair of a vertex subset $V' \subseteq V$ and an edge subset $E' \subseteq E$ of G, a graph $G' = (V', E')$ is called a* subgraph *of G.*

Definition 2.2.3. *For any edge subset $X \subseteq E$ of G, $V[X]$ denotes the* induced vertices *that is the set of endpoints of each edge in X, i.e., $V[X] := \bigcup_{\{u,v\} \in X} \{u,v\}$ or $V[X] := \bigcup_{(u,v) \in X} \{u,v\}$. Similarly, $G[X]$ denotes the* edge induced subgraph *such that $G[X] := (V[X], X)$.*

We deal with set families of edges on graphs for representing sets of edge induced subgraphs in our contributions.

Example 2.2.1. *Figure 2.1 and 2.2 show examples of graphs and their edge induced subgraphs. Each edge on undirected graph is drawn by a line segment, whereas each edge on directed graph is drawn by an arrow, whose direction is from u to v if the edge is (u,v).*

Definition 2.2.4. *An s-t* path *on G where $s,t \in V$ and $s \neq t$ is a sequence of distinct vertices $(s = v_1, v_2, \ldots, v_k = t)$ where $\{v_i, v_{i+1}\} \in E$ if G is undirected, otherwise $(v_i, v_{i+1}) \in E$, for any $i \in [k-1]$.*

(a) An undirected graph (b) Its edge induced subgraph

Figure 2.1. An undirected graph and its edge induced subgraph

(a) A directed graph (b) Its edge induced subgraph

Figure 2.2. A directed graph and its edge induced subgraph

2.3 Decision Diagrams

In this thesis, we use two data structures the BDD [15] and the ZDD [47] for representing set families compactly. The BDD is a compact graph representation of Boolean functions which are indicator functions of set families. The ZDD is a derivation of the BDD with a special rule for representing set families.

Here, we discuss on a set family $\mathcal{X}$ of a set $S = \{s_1, \ldots, s_n\}$, i.e., $\mathcal{X} \subseteq 2^S$. The elements are ordered as $s_1 < \ldots < s_n$. Let $S^{<i} := \{s_1, \ldots, s_{i-1}\}$, $S^{\geq i} := \{s_i, \ldots, s_n\}$.

2.3.1 Binary Decision Diagrams (BDDs)

An n variable Boolean function has the form $f\colon \{0,1\}^n \to \{0,1\}$ and is an indicator function of a set family as follows. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ be a variable assignment of a Boolean function f which is an indicator function of $\mathcal{X}$. $\boldsymbol{x}$ corresponds to a subset $X \subseteq S$ such that $s_i \in \mathcal{X}$ iff $x_i = 1$ ($i \in [n]$). Then $X \in \mathcal{X}$ iff $f(\boldsymbol{x}) = 1$.

A BDD is a layered directed graph $\mathcal{B} = (N, A)$ with a *node* set N and an *arc* set A. [1] N has exactly one root node ρ in the most top layer and exactly two terminal nodes $\bot$ and $\top$ in the most bottom layer. Each non-terminal node $\alpha \in N$ has a layer label $\ell(\alpha) \in [n]$ (i.e., α is associated with $\ell(\alpha)$-th variable $x_{\ell(\alpha)}$), and has exactly two outgoing arcs called 0-arc and 1-arc. The node pointed by b-arc of α is called b-child of α for each $b \in \{0,1\}$, which is denoted by α_b where $\ell(\alpha) < \ell(\alpha_b)$ if α_b is not a terminal.

A BDD $\mathcal{B}$ represents a Boolean function f which is an indicator function of $\mathcal{X}$ as follows: Each directed path from ρ to $\top$ represents a (possibly partial) variable assignment $\boldsymbol{x}$ for which $f(\boldsymbol{x}) = 1$. If a path descends a b-arc of a node α, a variable $x_{\ell(\alpha)}$ is assigned to b. Not assigned variables are *don't-care*, i.e., any assignment of them does not change the result $f(\boldsymbol{x}) = 1$. For each node $\alpha \in N$, let $\mathcal{B}(\alpha)$ be the set family that has any subset $X \cup S^{<\ell(\alpha)}$ where X is represented by a path from ρ to $\top$ descending α. Then $\mathcal{B}(\rho) = \mathcal{X}$, $\mathcal{B}(\top) = 2^S$, and $\mathcal{B}(\bot) = \emptyset$.

Although a most naive BDD forms a binary tree such as Figure 2.3a, it has $2^n - 1$ non-terminal nodes. For eliminating redundant nodes in BDDs, the following reduction rules are applied as long as possible.

- *Node deletion*: Delete a node α if $\alpha_0 = \alpha_1$. When α is deleted, each arc pointing α change its endpoint into the child of α. (Figure 2.4a)
- *Node sharing*: Share two nodes β and β' where if $\ell(\beta) = \ell(\beta')$, $\beta_b = \beta'_b$ for each $b \in \{0,1\}$. (Figure 2.4b)

Any BDD becomes the unique *reduced* form by applying the reduction rules (Figure 2.3b). The BDD size is often evaluated by the number of non-terminal nodes $|N| - 2$. For convenience, we write $|\beta|$ instead of $|N| - 2$.

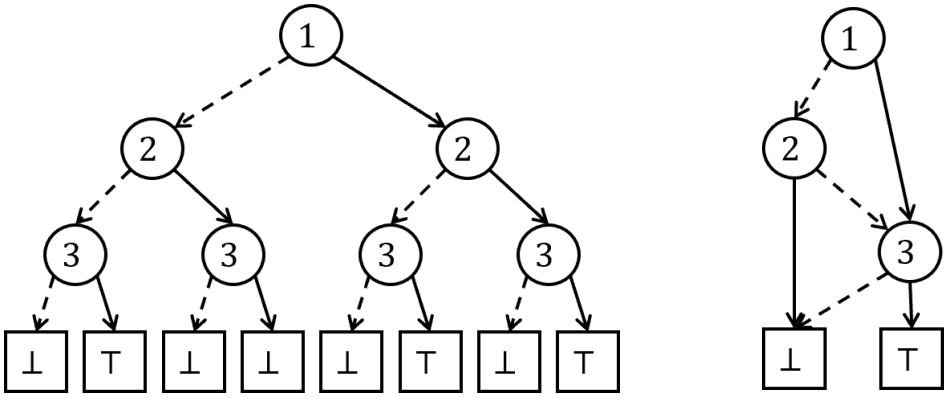
2.3.2 Zero-suppressed Binary Decision Diagrams (ZDDs)

A ZDD is also a layered directed graph $\mathcal{Z} = (N, A)$ as with a BDD. Primary differences between the BDD and the ZDD are in their representation rules of set families and their node deletion rules.

$\mathcal{Z}$ represents a set family $\mathcal{X}$ as follows: Each directed path from ρ to $\top$ represents a subset $X \in \mathcal{X}$. If a path descends an 1-arc of α, then $s_{\ell(\alpha)} \in X$, otherwise not so. For each node $\alpha \in N$, let $\mathcal{Z}(\alpha)$ be the set family that has any subset $X \cap S^{\geq \ell(\alpha)}$ where X is represented by a path from ρ to $\top$ descending α. Then $\mathcal{Z}(\rho) = \mathcal{X}$, $\mathcal{Z}(\top) = \{\emptyset\}$, and $\mathcal{Z}(\bot) = \emptyset$.

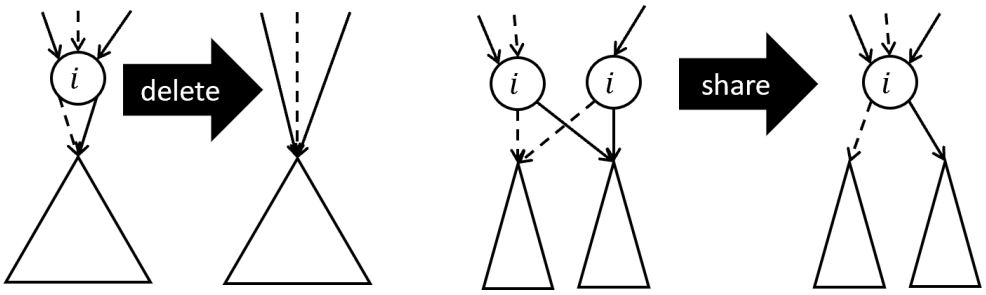
The node deletion rule of the ZDD is as follows:

[1] To avoid the confusion, we use the terms "vertex" and "edge" for a vertex and edge in the graph, and "node" and "arc" for a vertex and edge in the BDD. Vertices and nodes are denoted using Roman letters $(u, v, \ldots)$ and Greek letters $(\alpha, \beta, \ldots)$, respectively.



(a) A binary tree (b) A BDD

Figure 2.3. A binary tree and a BDD representing a Boolean function $f(\boldsymbol{x}) = x_1x_3 \vee \bar{x_1}\bar{x_2}x_3$ which is an indicator function of a set family $\mathcal{X} = \{\{s_1, s_3\}, \{s_1, s_2, s_3\}, \{s_3\}\}$



(a) Node deletion (b) Node sharing

Figure 2.4. Reduction rules of the BDD

- *Node deletion* (ZDD): Delete a node α if $\alpha_1 = \bot$. When α is deleted, each arc pointing α change its endpoint into the child of α. (Figure 2.5)

Node sharing rule does not differ between the BDD and the ZDD. Any ZDD also becomes the unique reduced form by applying the reduction rules (Figure 2.6). For convenience, we write the ZDD size of $\mathcal{Z}$ as $|\mathcal{Z}|$.

2.3.3 Variable Order

The *variable order* such as $s_1 < \ldots < s_n$ is a critical factor of the BDD/ZDD size changing. Even swapping two elements changes the BDD/ZDD size. Note that, although any set family has its optimal variable order which minimizes the BDD/ZDD size, the size does not always become small enough, i.e., $O(2^n)$.

There are some known results on the variable order: Improving the variable order is NP-complete [11]. Therefore, various heuristics for good ordering have been studied [3, 46]. Especially, on dealing with set families of graphs (i.e., S is an edge set or

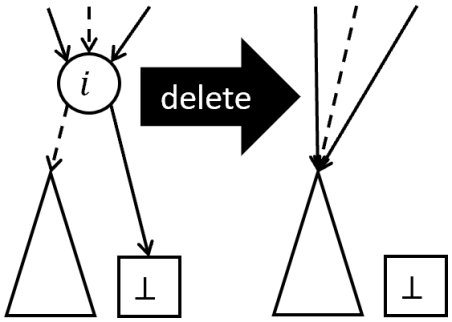


Figure 2.5. Node deletion of the ZDD

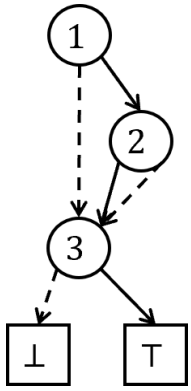


Figure 2.6. A ZDD representing a set family $\mathcal{X} = \{\{s_1, s_3\}, \{s_1, s_2, s_3\}, \{s_3\}\}$

a vertex set), the *path width* [39] leads a good upper bound of the BDD/ZDD size [28]. A heuristic algorithm based on the *path decomposition* finds a good variable order empirically [28].

2.4 Useful Algorithms on BDDs/ZDDs

The BDD/ZDD has some useful functions such as algebraic operations called the *family algebra* [40], counting the number of subsets $|\mathcal{B}(\rho)|$ and $|\mathcal{Z}(\rho)|$, computation of the occurrence probability of the elements in $\mathcal{B}(\rho)$ and $\mathcal{Z}(\rho)$, optimization with a linear criterion, and the FBS to construct the BDD/ZDD representing subgraphs. We introduce them in the following four subsections.

2.4.1 Family Algebra

The BDD supports various logical operations for Boolean functions [15, 40]. We introduce some operations on the BDD in Table 2.1. Note that f and g are the operands, and h is the result Boolean function. Let $\mathcal{B}_f$ and $\mathcal{B}_g$ be a BDD for f and g, respectively. As a known result, the operations $\vee$ and $\wedge$ can be conducted in the computation time $O(|\mathcal{B}_f||\mathcal{B}_g|)$ [67]. Operation $\neg$ takes $O(1)$ time because we should swap the $\bot$ and $\top$.

Table 2.1. Examples of the family algebra on the BDD.

Name	Operator	Formula	h
logical or	$\vee$	$f \vee g$	$h(\boldsymbol{x}) = 1 \iff f(\boldsymbol{x}) = 1$ or $g(\boldsymbol{x}) = 1$
logical and	$\wedge$	$f \wedge g$	$h(\boldsymbol{x}) = 1 \iff f(\boldsymbol{x}) = 1$ and $g(\boldsymbol{x}) = 1$
logical not	$\neg$	$\neg f$	$h(\boldsymbol{x}) = 1 \iff f(\boldsymbol{x}) = 0$

The ZDD supports various set operations including operations which are special to the ZDD [35, 40]. We introduce some operations on the ZDD in Table 2.2. Note that $\mathcal{F}$ and $\mathcal{G}$ are the operands, and $\mathcal{H}$ is the result set family. Let $\mathcal{Z}_\mathcal{F}$ and $\mathcal{Z}_\mathcal{G}$ be a ZDD for $\mathcal{F}$ and $\mathcal{G}$, respectively. As a known result, the operations $\cup$, $\cap$, and $\setminus$ can be conducted in the computation time $O(|\mathcal{Z}_f||\mathcal{Z}_g|)$ [67].

Table 2.2. Examples of the family algebra on the ZDD.

Name	Operator	Formula	$\mathcal{H}$
union	$\cup$	$\mathcal{F}\cup\mathcal{G}$	$\{X \mid X\in\mathcal{F} \text{ or } X\in\mathcal{G}\}$
intersection	$\cap$	$\mathcal{F}\cap\mathcal{G}$	$\{X \mid X\in\mathcal{F} \text{ and } X\in\mathcal{G}\}$
difference	$\setminus$	$\mathcal{F}\setminus\mathcal{G}$	$\{X \mid X\in\mathcal{F}, X\notin\mathcal{G}\}$
join	$\bowtie$	$\mathcal{F}\bowtie\mathcal{G}$	$\{X\cup Y \mid X\in\mathcal{F}, Y\in\mathcal{G}\}$
disjoint-join	$\overset{\cdot}{\bowtie}$	$\mathcal{F}\bowtie\mathcal{G}$	$\{X\cup Y \mid X\in\mathcal{F}, Y\in\mathcal{G}, X\cap Y=\emptyset\}$
joint-join	$\overset{+}{\bowtie}$	$\mathcal{F}\bowtie\mathcal{G}$	$\{X\cup Y \mid X\in\mathcal{F}, Y\in\mathcal{G}, X\cap Y\neq\emptyset\}$
restriction	$\lhd$	$\mathcal{F}\lhd\mathcal{G}$	$\{X \mid X\in\mathcal{F}, \exists Y\in\mathcal{G}, Y\subseteq X\}$

We omit the detail of algorithms for the family algebra.

2.4.2 Counting Number of Subsets

Given a set family $\mathcal{F}$ and an element e, let $\mathcal{F}\lhd e$ be a short notation of $\mathcal{F}\lhd\{\{e\}\}$, and $\bar{\lhd}$ be the binary operator where $\mathcal{F}\bar{\lhd}e := \{F\in\mathcal{F} \mid e\notin F\}$. Similarly, let $\mathcal{F}\bowtie e$ be a short notation of $\mathcal{F}\bowtie\{\{e\}\}$. For any node α of a BDD $\mathcal{B}$ or a ZDD $\mathcal{Z}$, the following recursive formulas are well-defined:

$$\mathcal{B}(\alpha)=\begin{cases}\emptyset & (\alpha=\bot),\\ 2^S & (\alpha=\top),\\ (\mathcal{B}(\alpha_0)\bar{\lhd}s_{\ell(\alpha)})\cup(\mathcal{B}(\alpha_1)\lhd s_{\ell(\alpha)}) & (\text{otherwise}).\end{cases} \tag{2.1}$$

$$\mathcal{Z}(\alpha)=\begin{cases}\emptyset & (\alpha=\bot),\\ \{\emptyset\} & (\alpha=\top),\\ \mathcal{Z}(\alpha_0)\cup(\mathcal{Z}(\alpha_1)\bowtie s_{\ell(\alpha)}) & (\text{otherwise}).\end{cases} \tag{2.2}$$

Here, we present an algorithm to compute $|\mathcal{B}(\rho)|$ and $|\mathcal{Z}(\rho)|$. Obviously, we can

obtain the following recursive formulas derived from (2.1) and (2.2):

$$|\mathcal{B}(\alpha)| = \begin{cases} 0 & (\alpha = \bot), \\ 2^n & (\alpha = \top), \\ \frac{|\mathcal{B}(\alpha_0)|+|\mathcal{B}(\alpha_1)|}{2} & (\text{otherwise}). \end{cases} \tag{2.3}$$

$$|\mathcal{Z}(\alpha)| = \begin{cases} 0 & (\alpha = \bot), \\ 1 & (\alpha = \top), \\ |\mathcal{Z}(\alpha_0)| + |\mathcal{Z}(\alpha_1)| & (\text{otherwise}). \end{cases} \tag{2.4}$$

They yield a dynamic programming algorithm as follows: Each node α of a BDD $\mathcal{B}$ stores a value $\Gamma_{\mathcal{B}}(\alpha)$ which is equal to $|\mathcal{B}(\alpha)|$. Similarly, each node α of a ZDD $\mathcal{Z}$ stores a value $\Gamma_{\mathcal{Z}}(\alpha)$ which is equal to $|\mathcal{Z}(\alpha)|$. Let $N_i := \{\alpha \mid \ell(\alpha) = i\}$ for each $i \in [n]$. First, we initialize $\Gamma_{\mathcal{B}}$ and $\Gamma_{\mathcal{Z}}$ as $\Gamma_{\mathcal{B}}(\bot) = 0$, $\Gamma_{\mathcal{B}}(\top) = 2^n$, $\Gamma_{\mathcal{Z}}(\bot) = 0$, and $\Gamma_{\mathcal{Z}}(\top) = 1$. Subsequently, computation of $\Gamma_{\mathcal{B}}$ and $\Gamma_{\mathcal{Z}}$ for all node can be conducted by the bottom-up processes from N_n to N_1 according to (2.3) and (2.4). Finally, $\Gamma_{\mathcal{B}}(\rho) = |\mathcal{B}(\rho)|$ and $\Gamma_{\mathcal{Z}}(\rho) = |\mathcal{Z}(\rho)|$. The algorithm takes $O(|\mathcal{B}|)$ and $O(|\mathcal{Z}|)$ time, respectively. The pseudo codes of the algorithm are shown in Algorithm 2.4.1 and 2.4.2.

Algorithm 2.4.1 Computing $|\mathcal{B}(\rho)|$

```
Γ_B(⊥) ← 0, Γ_B(⊤) ← 2^n
for i = n, ..., 1 do
    for α ∈ N_i do
        Γ_B(α) ← (Γ_B(α_0) + Γ_B(α_1)) / 2
    end for
end for
return Γ_B(ρ)
```

Algorithm 2.4.2 Computing $|\mathcal{Z}(\rho)|$

```
Γ_Z(⊥) ← 0, Γ_Z(⊤) ← 1
for i = n, ..., 1 do
    for α ∈ N_i do
        Γ_Z(α) ← Γ_Z(α_0) + Γ_Z(α_1)
    end for
end for
return Γ_Z(ρ)
```

2.4.3 Computation of Occurrence Probability

Given a probability function $p\colon S \to [0,1]$ where each element $s_i \in S$ independently drops from S in the probability $p(s_i)$, the occurrence probability of $\mathcal{X}$ is defined as follows:

$$\theta(\mathcal{X}) := \sum_{X \in \mathcal{X}} p(X) \tag{2.5}$$

where

$$p(X) := \prod_{s_j \in S \setminus X} p(s_j) \prod_{s_i \in X} (1 - p(s_i)). \tag{2.6}$$

We present an algorithm to compute $\theta(\mathcal{B}(\rho))$ and $\theta(\mathcal{Z}(\rho))$.

We can obtain the following recursive formulas derived from (2.1) and (2.2):

$$\theta(\mathcal{B}(\alpha)) = \begin{cases} 0 & (\alpha = \bot), \\ 1 & (\alpha = \top), \\ \theta(\mathcal{B}(\alpha_0))p(s_{\ell(\alpha)}) + \theta(\mathcal{B}(\alpha_1))(1 - p(s_{\ell(\alpha)})) & (\text{otherwise}). \end{cases} \tag{2.7}$$

$$\theta(\mathcal{Z}(\alpha)) = \begin{cases} 0 & (\alpha = \bot), \\ p(\emptyset) & (\alpha = \top), \\ \theta(\mathcal{Z}(\alpha_0)) + \theta(\mathcal{Z}(\alpha_1))(1 - 1/p(s_{\ell(\alpha)})) & (\text{otherwise}). \end{cases} \tag{2.8}$$

They yield a dynamic programming algorithm as follows: Each node α of a BDD $\mathcal{B}$ stores a value $\theta_{\mathcal{B}}(\alpha)$ which is equal to $\theta(\mathcal{B}(\alpha))$. Similarly, each node α of a ZDD $\mathcal{Z}$ stores a value $\theta_{\mathcal{Z}}(\alpha)$ which is equal to $\theta(\mathcal{Z}(\alpha))$. First, we initialize $\theta_{\mathcal{B}}$ and $\theta_{\mathcal{Z}}$ as $\theta_{\mathcal{B}}(\bot) = 0$, $\theta_{\mathcal{B}}(\top) = 1$, $\theta_{\mathcal{Z}}(\bot) = 0$, and $\theta_{\mathcal{Z}}(\top) = p(\emptyset)$. Subsequently, computation of $\theta_{\mathcal{B}}$ and $\theta_{\mathcal{Z}}$ for all node can be conducted by the bottom-up processes from N_n to N_1 according to (2.7) and (2.8). Finally, $\theta_{\mathcal{B}}(\rho) = \theta(\mathcal{B}(\rho))$ and $\theta_{\mathcal{Z}}(\rho) = \theta(\mathcal{Z}(\rho))$. The algorithm takes $O(|\mathcal{B}|)$ and $O(|\mathcal{Z}|)$ time, respectively. The pseudo codes of the algorithm are shown in Algorithm 2.4.3 and 2.4.4.

2.4.4 Optimization with A Linear Criterion

Let $c\colon S \to \mathbb{N}$ be a cost function where each element $s_i \in S$ has a cost $c(s_i)$. We are interested in the linear criteria represented by

$$c(X) := \sum_{s_i \in X} c(s_i). \tag{2.9}$$

Algorithm 2.4.3 Computing $\theta(\mathcal{B}(\rho))$

$\theta_{\mathcal{B}}(\bot) \leftarrow 0, \theta_{\mathcal{B}}(\top) \leftarrow 1$
for $i = n, \ldots, 1$ **do**
  for $\alpha \in N_i$ **do**
    $\theta_{\mathcal{B}}(\alpha) \leftarrow \theta_{\mathcal{B}}(\alpha_0)p(s_{\ell(\alpha)}) + \theta_{\mathcal{B}}(\alpha_1)(1 - p(s_{\ell(\alpha)}))$
  end for
end for
return $\theta_{\mathcal{B}}(\rho)$

Algorithm 2.4.4 Computing $\theta(\mathcal{Z}(\rho))$

$\theta_{\mathcal{Z}}(\bot) \leftarrow 0, \theta_{\mathcal{Z}}(\top) \leftarrow p(\emptyset)$
for $i = n, \ldots, 1$ **do**
  for $\alpha \in N_i$ **do**
    $\theta_{\mathcal{Z}}(\alpha) \leftarrow \theta_{\mathcal{Z}}(\alpha_0) + \theta_{\mathcal{Z}}(\alpha_1)(1 - 1/p(s_{\ell(\alpha)}))$
  end for
end for
return $\theta_{\mathcal{Z}}(\rho)$

In the following, we present an algorithm to find an optimal subset $X^* \in \mathcal{X}$ that minimizes/maximizes c, i.e.,

$$X^* \in \arg\min_{X \in \mathcal{X}} c(X) \tag{2.10}$$

or

$$X^* \in \arg\max_{X \in \mathcal{X}} c(X) \tag{2.11}$$

where $\mathcal{X} = \mathcal{B}(\rho)$ or $\mathcal{X} = \mathcal{Z}(\rho)$.

Minimization

Let $\Psi(\mathcal{B}(\alpha))$ and $\Psi(\mathcal{Z}(\alpha))$ be the minimum value on $\mathcal{B}(\alpha)$ and $\mathcal{Z}(\alpha)$, respectively: Namely, $\Psi(\mathcal{B}(\alpha)) := \min\{c(X) \mid X \in \mathcal{B}(\alpha)\}$ and $\Psi(\mathcal{Z}(\alpha)) := \min\{c(X) \mid X \in \mathcal{Z}(\alpha)\}$. Note that $\Psi(\mathcal{B}(\rho)) = c(X^*)$ and $\Psi(\mathcal{Z}(\rho)) = c(X^*)$. We can obtain the following recur-

sive formulas derived from (2.1) and (2.2):

$$\Psi(\mathcal{B}(\alpha)) = \begin{cases} \infty & (\alpha = \bot), \\ 0 & (\alpha = \top), \\ \min\{\Psi(\mathcal{B}(\alpha_0)), \Psi(\mathcal{B}(\alpha_1)) + c(s_{\ell(\alpha)})\} & (\text{otherwise}). \end{cases} \tag{2.12}$$

$$\Psi(\mathcal{Z}(\alpha)) = \begin{cases} \infty & (\alpha = \bot), \\ 0 & (\alpha = \top), \\ \min\{\Psi(\mathcal{Z}(\alpha_0)), \Psi(\mathcal{Z}(\alpha_1)) + c(s_{\ell(\alpha)})\} & (\text{otherwise}). \end{cases} \tag{2.13}$$

They yield a dynamic programming algorithm as follows: Each node α of a BDD $\mathcal{B}$ stores a value $\Psi_{\mathcal{B}}(\alpha)$ which is equal to $\Psi(\mathcal{B}(\alpha))$. Similarly, each node α of a ZDD $\mathcal{Z}$ stores a value $\Psi_{\mathcal{Z}}(\alpha)$ which is equal to $\Psi(\mathcal{Z}(\alpha))$. First, we initialize $\Psi_{\mathcal{B}}$ and $\Psi_{\mathcal{Z}}$ as $\Psi_{\mathcal{B}}(\bot) = \infty$, $\Psi_{\mathcal{B}}(\top) = 0$, $\Psi_{\mathcal{Z}}(\bot) = \infty$, and $\Psi_{\mathcal{Z}}(\top) = 0$. Subsequently, computation of $\Psi_{\mathcal{B}}$ and $\Psi_{\mathcal{Z}}$ for all node can be conducted by the bottom-up processes from N_n to N_1 according to (2.12) and (2.13). Finally, $\Psi_{\mathcal{B}}(\rho) = \Psi(\mathcal{B}(\rho))$ and $\Psi_{\mathcal{Z}}(\rho) = \Psi(\mathcal{Z}(\rho))$. The algorithm takes $O(|\mathcal{B}|)$ and $O(|\mathcal{Z}|)$ time, respectively. The pseudo codes of the algorithm are shown in Algorithm 2.4.5 and 2.4.6.

Algorithm 2.4.5 Computing $c(X^*)$ where $X^* \in \arg\min_{X \in \mathcal{B}(\rho)} c(X)$

```
1: Ψ_B(⊥) ← ∞, Ψ_B(⊤) ← 0
2: for i = n, ..., 1 do
3:     for α ∈ N_i do
4:         Ψ_B(α) ← min{Ψ_B(α_0), Ψ_B(α_1) + c(s_ℓ(α))}
5:     end for
6: end for
7: return Ψ_B(ρ)
```

Algorithm 2.4.6 Computing $c(X^*)$ where $X^* \in \arg\min_{X \in \mathcal{Z}(\rho)} c(X)$

```
1: Ψ_Z(⊥) ← ∞, Ψ_Z(⊤) ← 0
2: for i = n, ..., 1 do
3:     for α ∈ N_i do
4:         Ψ_Z(α) ← min{Ψ_Z(α_0), Ψ_Z(α_1) + c(s_ℓ(α))}
5:     end for
6: end for
7: return Ψ_Z(ρ)
```

For extracting an optimal subset X^* from $\mathcal{B}$ and $\mathcal{Z}$, we use the computed value $\Psi_{\mathcal{B}}$ and $\Psi_{\mathcal{Z}}$: We descend the BDD $\mathcal{B}$ (resp. the ZDD $\mathcal{Z}$) starting at the root node ρ and ending at the terminal node $\top$. Let β be a current node. Initially, we have an empty set X. We repeat the following process while $\beta \neq \top$. We select $b \in \{0,1\}$ where $\Psi_{\mathcal{B}}(\beta) = \Psi_{\mathcal{B}}(\beta_b) + b \times c(s_{\ell(\beta)})$ (resp. $\Psi_{\mathcal{Z}}(\beta) = \Psi_{\mathcal{Z}}(\beta_b) + b \times c(s_{\ell(\beta)})$), and descend to β_b. Simultaneously, if $b = 1$, we add $s_{\ell(\beta)}$ to X. Finally, we have $X = X^*$. Because the descending process finish in at most n steps, the computation time is $O(n)$. The pseudo codes of the extracting process are shown in Algorithm 2.4.7 and 2.4.8.

Algorithm 2.4.7 Extract a subset $X^* \in \arg\min_{X \in \mathcal{B}(\rho)} c(X)$

1: $\beta \leftarrow \rho, X \leftarrow \emptyset$
   {Assuming that the Algorithm 2.4.5 ran on $\mathcal{B}$}
2: **while** $\beta \neq \top$ **do**
3: Select $b \in \{0,1\}$ where $\Psi_{\mathcal{B}}(\beta) = \Psi_{\mathcal{B}}(\beta_b) + b \times c(s_{\ell(\beta)})$
4: **if** $b = 1$ **then**
5: $X \leftarrow X \cup \{s_{\ell(\beta)}\}$
6: **end if**
7: $\beta \leftarrow \beta_b$
8: **end while**
9: **return** X

Algorithm 2.4.8 Extract a subset $X^* \in \arg\min_{X \in \mathcal{Z}(\rho)} c(X)$

1: $\beta \leftarrow \rho, X \leftarrow \emptyset$
   {Assuming that the Algorithm 2.4.6 ran on $\mathcal{Z}$}
2: **while** $\beta \neq \top$ **do**
3: Select $b \in \{0,1\}$ where $\Psi_{\mathcal{Z}}(\beta) = \Psi_{\mathcal{Z}}(\beta_b) + b \times c(s_{\ell(\beta)})$
4: **if** $b = 1$ **then**
5: $X \leftarrow X \cup \{s_{\ell(\beta)}\}$
6: **end if**
7: $\beta \leftarrow \beta_b$
8: **end while**
9: **return** X

Maximization

Let $\Upsilon(\mathcal{B}(\alpha))$ and $\Upsilon(\mathcal{Z}(\alpha))$ be the maximum value on $\mathcal{B}(\alpha)$ and $\mathcal{Z}(\alpha)$, respectively: Namely, $\Upsilon(\mathcal{B}(\alpha)) := \max\{c(X) \mid X \in \mathcal{B}(\alpha)\}$ and $\Upsilon(\mathcal{Z}(\alpha)) := \max\{c(X) \mid X \in \mathcal{Z}(\alpha)\}$.

Note that $\Upsilon(\mathcal{B}(\rho)) = c(X^*)$ and $\Upsilon(\mathcal{Z}(\rho)) = c(X^*)$. We can obtain the following recursive formulas derived from (2.1) and (2.2):

$$\Upsilon(\mathcal{B}(\alpha)) = \begin{cases} -\infty & (\alpha = \bot), \\ c(S) & (\alpha = \top), \\ \max\{\Upsilon(\mathcal{B}(\alpha_0)) - c(s_{\ell(\alpha)}), \Upsilon(\mathcal{B}(\alpha_1))\} & (\text{otherwise}). \end{cases} \tag{2.14}$$

$$\Upsilon(\mathcal{Z}(\alpha)) = \begin{cases} -\infty & (\alpha = \bot), \\ 0 & (\alpha = \top), \\ \max\{\Upsilon(\mathcal{Z}(\alpha_0)), \Upsilon(\mathcal{Z}(\alpha_1)) + c(s_{\ell(\alpha)})\} & (\text{otherwise}). \end{cases} \tag{2.15}$$

They yield a dynamic programming algorithm as follows: Each node α of a BDD $\mathcal{B}$ stores a value $\Upsilon_{\mathcal{B}}(\alpha)$ which is equal to $\Upsilon(\mathcal{B}(\alpha))$. Similarly, each node α of a ZDD $\mathcal{Z}$ stores a value $\Upsilon_{\mathcal{Z}}(\alpha)$ which is equal to $\Upsilon(\mathcal{Z}(\alpha))$. First, we initialize $\Upsilon_{\mathcal{B}}$ and $\Upsilon_{\mathcal{Z}}$ as $\Upsilon_{\mathcal{B}}(\bot) = -\infty$, $\Upsilon_{\mathcal{B}}(\top) = c(S)$, $\Upsilon_{\mathcal{Z}}(\bot) = -\infty$, and $\Upsilon_{\mathcal{Z}}(\top) = 0$. Subsequently, computation of $\Upsilon_{\mathcal{B}}$ and $\Upsilon_{\mathcal{Z}}$ for all node can be conducted by the bottom-up processes from N_n to N_1 according to (2.14) and (2.15). Finally, $\Upsilon_{\mathcal{B}}(\rho) = \Upsilon(\mathcal{B}(\rho))$ and $\Upsilon_{\mathcal{Z}}(\rho) = \Upsilon(\mathcal{Z}(\rho))$. The algorithm takes $O(|\mathcal{B}|)$ and $O(|\mathcal{Z}|)$ time, respectively. The pseudo codes of the algorithm are shown in Algorithm 2.4.9 and 2.4.10.

Algorithm 2.4.9 Computing $c(X^*)$ where $X^* \in \arg\max_{X \in \mathcal{B}(\rho)} c(X)$

```
1: Υ_B(⊥) ← −∞, Υ_B(⊤) ← c(S)
2: for i = n, ..., 1 do
3:    for α ∈ N_i do
4:       Υ_B(α) ← max{Υ_B(α_0) − c(s_ℓ(α)), Υ_B(α_1)}
5:    end for
6: end for
7: return Υ_B(ρ)
```

For extracting an optimal subset X^* from $\mathcal{B}$ and $\mathcal{Z}$, we use the computed value $\Upsilon_{\mathcal{B}}$ and $\Upsilon_{\mathcal{Z}}$: We descend the BDD $\mathcal{B}$ (resp. the ZDD $\mathcal{Z}$) starting at the root node ρ and ending at the terminal node $\top$. Let β be a current node. Initially, we have a set $X = S$ (resp. an empty set X). We repeat the following process while $\beta \neq \top$. We select $b \in \{0,1\}$ where $\Upsilon_{\mathcal{B}}(\beta) = \Upsilon_{\mathcal{B}}(\beta_b) - (1-b) \times c(s_{\ell(\beta)})$ (resp. $\Upsilon_{\mathcal{Z}}(\beta) = \Upsilon_{\mathcal{Z}}(\beta_b) + b \times c(s_{\ell(\beta)})$), and descend to β_b. Simultaneously, if $b = 0$, we delete $s_{\ell(\beta)}$ from X (resp. if $b = 1$, we add $s_{\ell(\beta)}$ to X). Finally, we have $X = X^*$. Because the descending process finish in at most n steps, the computation time is $O(n)$. The pseudo codes of the extracting process are shown in Algorithm 2.4.11 and 2.4.12.

Algorithm 2.4.10 Computing $c(X^*)$ where $X^* \in \arg\max_{X \in \mathcal{Z}(\rho)} c(X)$

1: $\Upsilon_{\mathcal{Z}}(\bot) \leftarrow -\infty, \Upsilon_{\mathcal{Z}}(\top) \leftarrow 0$
2: **for** $i = n, \ldots, 1$ **do**
3: **for** $\alpha \in N_i$ **do**
4: $\Upsilon_{\mathcal{Z}}(\alpha) \leftarrow \max\{\Upsilon_{\mathcal{Z}}(\alpha_0), \Upsilon_{\mathcal{Z}}(\alpha_1) + c(s_{\ell(\alpha)})\}$
5: **end for**
6: **end for**
7: **return** $\Upsilon_{\mathcal{Z}}(\rho)$

Algorithm 2.4.11 Extract a subset $X^* \in \arg\max_{X \in \mathcal{B}(\rho)} c(X)$

1: $\beta \leftarrow \rho, X \leftarrow S$
 {Assuming that the Algorithm 2.4.9 ran on $\mathcal{B}$}
2: **while** $\beta \neq \top$ **do**
3: Select $b \in \{0,1\}$ where $\Upsilon_{\mathcal{B}}(\beta) = \Upsilon_{\mathcal{B}}(\beta_b) - (1-b) \times c(s_{\ell(\beta)})$
4: **if** $b = 0$ **then**
5: $X \leftarrow X \setminus \{s_{\ell(\beta)}\}$
6: **end if**
7: $\beta \leftarrow \beta_b$
8: **end while**
9: **return** X

2.4.5 Frontier-Based Search

Here, we briefly introduce the FBS, which is a framework for top-down construction of decision diagrams representing subgraphs of a given graph. Let a given graph be $G = (V, E)$ where $E = \{e_1, \ldots, e_n\}$, $E^{<i} := \{e_1, \ldots, e_{i-1}\}$, and $E^{\geq i} := \{e_i, \ldots, e_n\}$.

Basically, the FBS constructs a binary tree in top-down manner. The algorithm processes the edges from e_1 to e_m and expands child nodes from ρ to $\bot$ and $\top$ by the exclusion/inclusion of each edge. A core of the FBS is to prune and merge redundant nodes like the reduction rules of the BDD/ZDD without expanding their child nodes. Thus, how to efficiently prune and merge redundant nodes is important in the FBS.

On processing an edge e_i, the FBS focuses on the local structure called the i-th *frontier* that is defined as:

$$F_i := V[E^{<i}] \cap V[E^{\geq i}]. \tag{2.16}$$

F_i is a set of all the vertices on both of the processed and the unprocessed edges. Then, each node associated with e_i stores an information called the *mate*, which localizes the components on F_i made by the processed edges. The mate must have the condition that

Algorithm 2.4.12 Extract a subset $X^* \in \arg\max_{X \in \mathcal{Z}(\rho)} c(X)$

1: $\beta \leftarrow \rho, X \leftarrow \emptyset$
   {Assuming that the Algorithm 2.4.6 ran on $\mathcal{Z}$}
2: **while** $\beta \neq \top$ **do**
3: Select $b \in \{0,1\}$ where $\Upsilon_{\mathcal{Z}}(\beta) = \Upsilon_{\mathcal{Z}}(\beta_b) + b \times c(s_{\ell(\beta)})$
4: **if** $b = 1$ **then**
5: $X \leftarrow X \cup \{s_{\ell(\beta)}\}$
6: **end if**
7: $\beta \leftarrow \beta_b$
8: **end while**
9: **return** X

if two nodes store same mate then they can be merged, i.e., they have same solutions on unprocessed edges. In addition, pruning condition should be defined by the mate. For example, for *s-t* path enumeration, a mate represents a set of path matching like the Figure 2.7. The Figure 2.8 shows examples of the pruning condition defined by the mate. Finally, we shows a brief sketch of the FBS in the Figure 2.9.

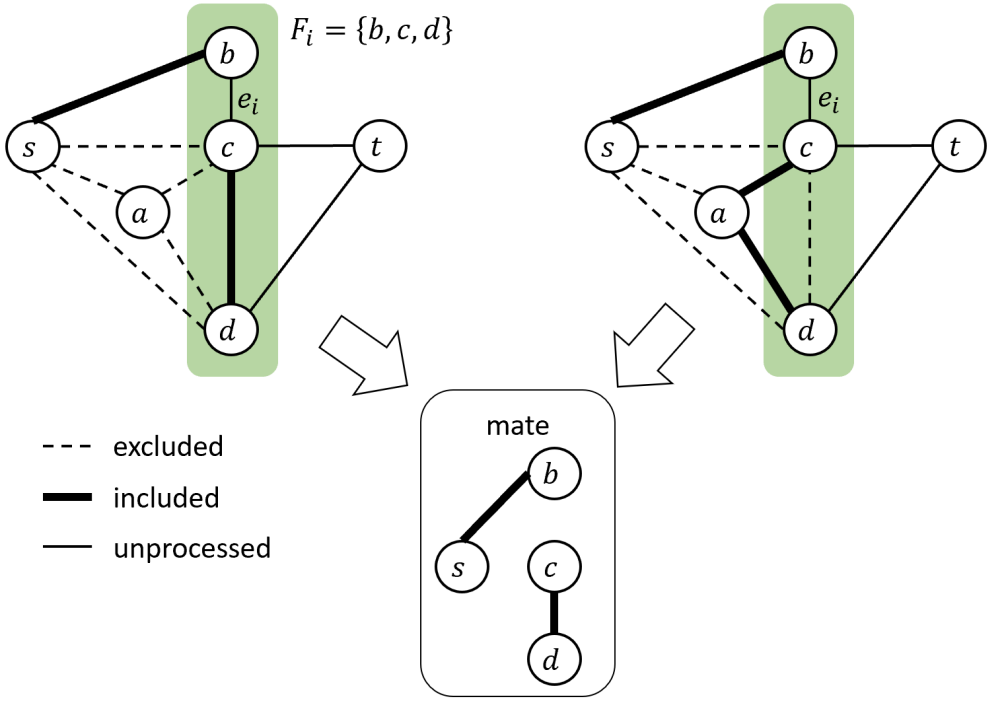


Figure 2.7. Example mate of the FBS for s-t path enumeration. The i-th frontier is $F_i = \{b, c, d\}$. Two subgraphs are localized as two path matching s-b and c-d. They can be merged, because an s-t paths is completed by adding the edge $\{b, c\}$ and $\{d, t\}$ in both subgraphs.

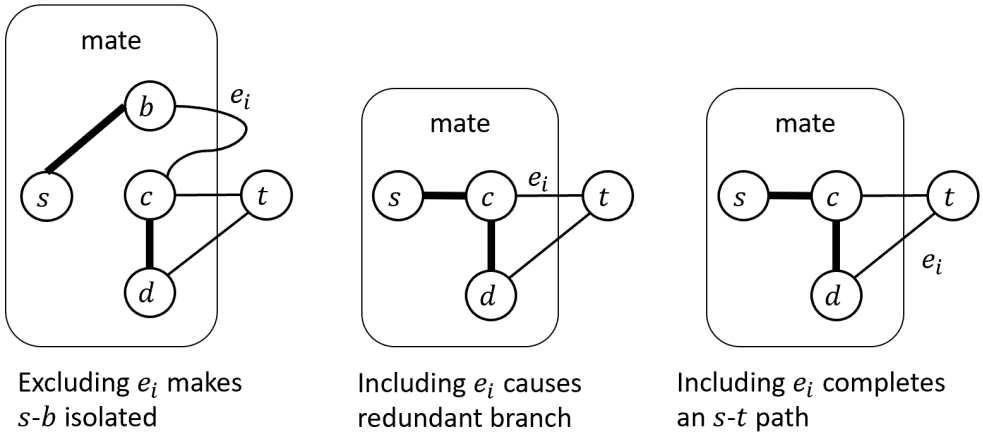


Figure 2.8. Example pruning condition which can be detected by the mate. In the left case and the center case, the child node becomes $\bot$, because there is no s-t path. In the right case, the child node becomes $\top$, because an s-t path is completed.

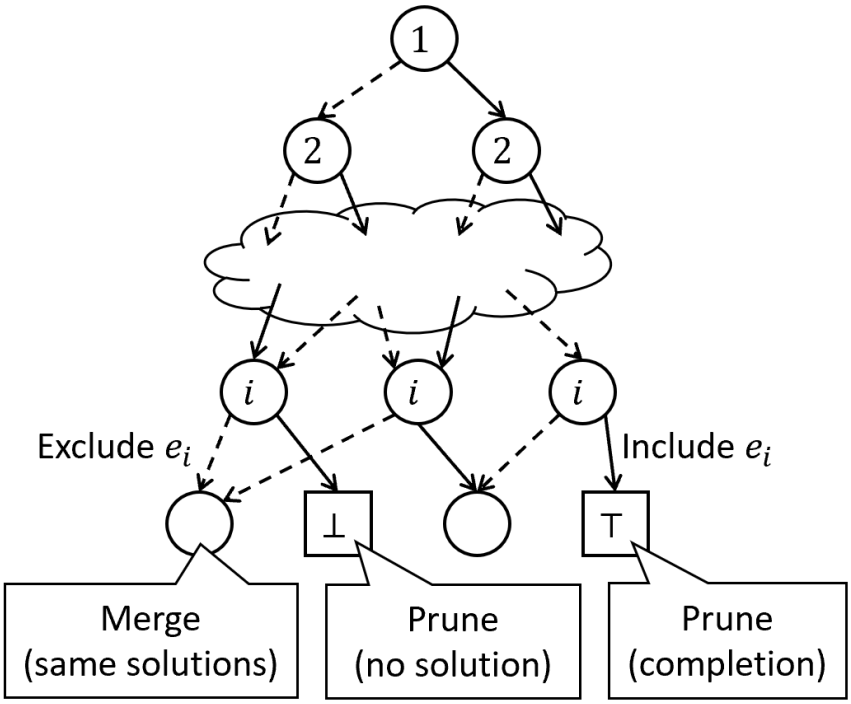


Figure 2.9. Brief sketch of the FBS

Chapter 3

Implicit Subset Enumeration by Top-Down Construction of DDs

In this chapter, we introduce the *subset enumeration problem* (SEP) that is the main target problem of this thesis. Subsequently, we formulate the key framework TD-DD by generalizing the FBS mentioned in the above chapter. The TD-DD conducts a top-down construction of decision diagrams to solve various SEPs efficiently, whereas the FBS deal with only the SEPs of graph structures.

3.1 Subset Enumeration Problem

Given a set S and a property function $C\colon 2^S \to \{0,1\}$, if $C(X) = 1$ for a subset $X \in 2^S$, then we say that X has the property C. An SEP $P = \langle S, C \rangle$ requires obtaining the set family $\mathcal{X}_P \subseteq 2^S$ where any subset $X \in \mathcal{X}_P$ has the property C:

$$\mathcal{X}_P := \{X \in 2^S \mid C(X) = 1\}. \tag{3.1}$$

In general, because the number of possible subsets is exponential in $|S|$, we should avoid the explicit enumeration for solving SEPs.

In the following section, we present the framework of the TD-DD for solving SEPs, which achieves an implicit enumeration: Given an SEP P, the algorithm constructs a BDD/ZDD $\mathcal{D}$ where $\mathcal{D}(\rho) = \mathcal{X}_P$ with a top-down manner. As with the previous chapter, we use the notations such that $n = |S|$, $S = \{s_1, \ldots, s_n\}$ where $s_1 < \ldots < s_n$, $S^{<i} := \{s_1, \ldots, s_{i-1}\}$, and $S^{\geq i} := \{s_i, \ldots, s_n\}$.

3.2 Framework of TD-DD

Let $\mathcal{D} = (N, A)$ be an initial decision diagram such that $N = \{\rho, \bot, \top\}$ where $\ell(\rho) = 1$ and $A = \emptyset$. Given an SEP P, the algorithm processes the elements of S as the exhaustive

search; the algorithm constructs the node set $N_i := \{\alpha \mid \ell(\alpha) = i\}$ for $i = 1, \ldots, n$ in this order, and the b-arc set $A_b := \{(\alpha, \alpha_b) \mid \exists i \in [n], \alpha \in N_i\}$ for each $b \in \{0,1\}$. For each node $\alpha \in N_i$, let $\bar{\mathcal{D}}(\alpha) \subseteq 2^{S^{<i}}$ be the set family that has any subset represented by a path from ρ to α according to the representation rule of the ZDD. Note that $\bar{\mathcal{D}}(\rho) = \{\emptyset\}$.

Each node $\alpha \in N_i$ is associated with a subproblem of P denoted by $P[\alpha] := \langle S^{\geq i}, C_\alpha \rangle$ where the property C_α is defined as

$$C_\alpha(X) = 1 \iff \forall Y \in \bar{\mathcal{D}}(\alpha),\ C(X \cup Y) = 1. \tag{3.2}$$

For any pair of nodes $\beta, \beta' \in N_i$, β and β' are *equivalent* if $C_\beta(X) = 1 \iff C_{\beta'}(X) = 1$ for any $X \in 2^{S^{\geq i}}$. The algorithm merges some equivalent nodes into one node.

The primary process of the algorithm is as follows. Initially, the algorithm generates the node set $N_1 = \{\rho\}$. At the i-th step, the algorithm constructs N_{i+1} using N_i as follows. For each node $\alpha \in N_i$, the algorithm generates its children; $\bar{\mathcal{D}}(\alpha_0)$ (resp. $\bar{\mathcal{D}}(\alpha_1)$) is the set family that s_i is excluded from (resp. included in) each subset of $\bar{\mathcal{D}}(\alpha)$. On generating a new child, the algorithm conducts the following procedures to reduce the number of nodes:

- $\bot$-*pruning*: Let $\bot$-prune(α, s_i, x) be the function, which returns a truth-value *True* or *False*, defined as follows: If $i < n$,

  $$\bot\text{-prune}(\alpha, s_i, b) = \mathit{True} \Longrightarrow \mathcal{X}_{P[\alpha_b]} = \emptyset. \tag{3.3}$$

  Otherwise ($i = n$),

  $$\bot\text{-prune}(\alpha, s_n, b) = \mathit{True} \iff \mathcal{X}_{P[\alpha_b]} = \emptyset. \tag{3.4}$$

  If $\bot$-prune(α, s_i, b) returns *True*, the b-child of α is $\bot$. Then the algorithm adds the b-arc $(\alpha, \bot)$ to A_b.

- $\top$-*pruning*: Let $\top$-prune(α, s_i, b) be the function, which returns a truth-value *True* or *False*, defined as follows: If $i < n$ and $\mathcal{D}$ becomes a BDD,

  $$\top\text{-prune}(\alpha, s_i, b) = \mathit{True} \Longrightarrow \mathcal{X}_{P[\alpha_b]} = 2^{S^{\geq i+1}}. \tag{3.5}$$

  If $i < n$ and $\mathcal{D}$ becomes a ZDD,

  $$\top\text{-prune}(\alpha, s_i, b) = \mathit{True} \Longrightarrow \mathcal{X}_{P[\alpha_b]} = \{\emptyset\}. \tag{3.6}$$

  If $i = n$,

  $$\top\text{-prune}(\alpha, s_n, b) = \mathit{True} \iff \mathcal{X}_{P[\alpha_b]} = \{\emptyset\}. \tag{3.7}$$

  If $\top$-prune(α, s_i, b) returns *True*, the b-child of α is $\top$. Then the algorithm adds the b-arc $(\alpha, \top)$ to A_b.

- *merging*: Let β be a child of α. If β and a node $\beta' \in N_{i+1}$ are equivalent, the algorithm sets β' to β.

To apply these procedures efficiently, each node β maintains an additional information $\phi(\beta)$, referred to as a *configuration* that satisfies the condition that

$$\phi(\beta) = \phi(\beta') \Rightarrow \beta \text{ and } \beta' \text{ are equivalent.} \tag{3.8}$$

Note that the inverse is not required, which causes redundant node expansions.

The general framework of the top-down construction is shown in Algorithm 3.2.1. The function generateNode(α, s_i, b) generates the b-child of α. Simultaneously, it computes the next configuration $\phi(\alpha_b)$. Note that, the constructed decision diagram is not necessarily reduced because redundant node expansions can be caused.

Algorithm 3.2.1 Top-down construction of a decision diagram for an SEP $P = \langle S, C \rangle$ with a generateNode function, a $\bot$-prune function, and a $\top$-prune function

$N_1 \leftarrow \{\rho\}$, $N_i \leftarrow \emptyset$ for $i = 2, \ldots, n$
Generate the terminals $\bot$ and $\top$.
$A_b \leftarrow \emptyset$ for each $b \in \{0,1\}$
for $i = 1, \ldots, n$ **do**
  for $\alpha \in N_i$ **do**
    for $b \in \{0,1\}$ **do**
      if $\bot$-prune$(\alpha, s_i, b) = True$ **then**
        $A_b \leftarrow A_b \cup \{(\alpha, \bot)\}$
      else if $\top$-prune$(\alpha, s_i, b) = True$ **then**
        $A_b \leftarrow A_b \cup \{(\alpha, \top)\}$
      else
        $\beta \leftarrow$ generateNode(α, s_i, b)
        if $\exists \beta' \in N_{i+1},\ \phi(\beta) = \phi(\beta')$ **then**
          $\beta \leftarrow \beta'$
        else
          $N_{i+1} \leftarrow N_{i+1} \cup \{\beta\}$
        end if
        $A_b \leftarrow A_b \cup \{(\alpha, \beta)\}$
      end if
    end for
  end for
end for
$N \leftarrow (\bigcup_{i=1,\ldots,m} N_i) \cup \{\bot, \top\}$, $A \leftarrow A_0 \cup A_1$
return $\mathcal{D} = (N, A)$

3.3 Examples

In this section, we introduce two examples of the TD-DD. One is the case that subsets are constrained by a knapsack-type constraint. The other is the case that subsets are equal to *degree constrained subgraphs* (DCSs) on undirected graphs. These examples are put into practice in some later chapters. We adapt the TD-DD to each SEP by designing *four primary components*: configuration, generateNode function, $\bot$-prune function, and $\top$-prune function.

3.3.1 Itemsets with Knapsack-Type Constraints

Let w be a weight function $w\colon S \to \mathbb{N}$ where $s_i \in S$ has the weight $w(s_i)$. Let $w(X)$ be the total weight of $X \subseteq S$: $w(X) := \sum_{s_i \in X} w(s_i)$. For a given capacity W, we design the four primary components to solve the SEP $P_{\mathrm{KP}} = \langle S, C_{\mathrm{KP}} \rangle$ where C_{KP} is the property function defined as

$$C_{\mathrm{KP}}(X) = 1 \iff w(X) \leq W. \tag{3.9}$$

Configuration

We use the current total weight as the configuration: for all $X \in \bar{\mathcal{D}}(\alpha)$,

$$\phi(\alpha) := w(X). \tag{3.10}$$

Here, we have the following lemma.

Lemma 3.3.1. *The configuration* (3.10) *satisfies the condition* (3.8).

Proof. For any node $\alpha \in N_i$, the following is satisfied by the definitions:

$$\mathcal{X}_{P_{\mathrm{KP}}[\alpha]} = \{X \in 2^{\geq i} \mid \phi(\alpha) + w(X) \leq W\}. \tag{3.11}$$

For any pair of nodes $\beta, \beta' \in N_i$, because (3.11) depends on only the configuration (3.10), we have

$$\phi(\beta) = \phi(\beta') \Rightarrow \mathcal{X}_{P_{\mathrm{KP}}[\beta]} = \mathcal{X}_{P_{\mathrm{KP}}[\beta']}. \tag{3.12}$$

□

generateNode Function

Initially, $\phi(\rho) = 0$ because $\bar{\mathcal{D}}(\rho) = \{\emptyset\}$. Subsequently, for any node $\alpha \in N_i$ and $b \in \{0,1\}$, generateNode(α, s_i, b) computes $\phi(\alpha_b) = \phi(\alpha) + b \times w(s_i)$.

$\bot$-prune Function

For any node $\alpha \in N_i$, because $\mathcal{X}_{P_{\mathrm{KP}}[\alpha_b]} = \emptyset$ iff $\phi(\alpha_b)$ is greater than the capacity W, we can design the $\bot$-prune function as follows:

$$\bot\text{-prune}(\alpha, s_i, b) := \begin{cases} \mathit{True} & (\phi(\alpha) + b \times w(s_i) > W), \\ \mathit{False} & (\text{otherwise}). \end{cases} \tag{3.13}$$

$\top$-prune Function

Because $\mathcal{X}_{P_{\mathrm{KP}}[\alpha_b]} = 2^{S^{\geq i+1}}$ iff all the remaining items $S^{\geq i}$ can be taken with the capacity $W - \phi(\alpha_b)$, we can design the $\top$-prune function for the BDD as follows:

$$\top\text{-prune}(\alpha, s_i, b) := \begin{cases} \mathit{True} & (W - (\phi(\alpha) + b \times w(s_i)) \geq w(S^{\geq i+1})), \\ \mathit{False} & (\text{otherwise}). \end{cases} \tag{3.14}$$

Let $w^*(S^{\geq i}) := \min\{w(s_j) \mid s_j \in S^{\geq i}\}$. Because $\mathcal{X}_{P_{\mathrm{KP}}[\alpha_b]} = \{\emptyset\}$ iff $W - \phi(\alpha_b)$ is less than $w^*(S^{\geq i+1})$, we can design the $\top$-prune function for the ZDD as follows:

$$\top\text{-prune}(\alpha, s_i, b) := \begin{cases} \mathit{True} & (W - (\phi(\alpha) + b \times w(s_i)) < w^*(S^{\geq i+1})), \\ \mathit{False} & (\text{otherwise}). \end{cases} \tag{3.15}$$

Then, we can solve the SEP P_{KP} by the TD-DD.

3.3.2 Degree Constrained Subgraphs

First, we introduce the following definition.

Definition 3.3.1. *On an undirected graph $G = (V, E)$, the* degree *of a vertex $v \in V$ is the number of edges having the endpoint v, which is denoted by $deg_G(v) := |\{e \in E \mid v \in e\}|$.*

Given a graph $G = (V, E)$, for any vertex $v \in V$, let $dc(v) \subseteq [0, |V| - 1]$ be a *degree constraint* for v. A subgraph $G[X]$ ($X \subseteq E$) is called an DCS if it satisfies the condition $deg_{G[X]}(v) \in dc(v)$ for all $v \in V$. For a given graph G and a degree constraint dc, we design the four primary components to solve the SEP $P_{\mathrm{DC}} = \langle E, C_{\mathrm{DC}} \rangle$ where C_{DC} is the property function defined as

$$C_{\mathrm{DC}}(X) = 1 \iff \forall v \in V, deg_{G[X]}(v) \in dc(v). \tag{3.16}$$

Let $E := \{e_1, \ldots, e_n\}$, $E^{<i} := \{e_1, \ldots, e_{i-1}\}$, and $E^{\geq i} := \{e_i, \ldots, e_n\}$ for convenience.

Configuration

We use the current degree of the frontier F_i as the configuration: for all $v \in F_i$ and all $X \in \bar{\mathcal{D}}(\alpha)$,

$$\phi_v(\alpha) := deg_{G[X]}(v). \tag{3.17}$$

For convenience, we write $\phi_v(\alpha) = 0$ if $v \notin F_i$. Let $\bar{F}_i$ be the set of vertices which leave the frontier before the i-th step: $\bar{F}_i := V[E^{<i}] \setminus F_i$. Here, we have the following lemma.

Lemma 3.3.2. *Assuming that any vertex $v \in \bar{F}_i$ satisfies $deg_{G[X]}(v) \in dc(v)$ ($X \in \bar{\mathcal{D}}(\alpha)$), then the configuration* (3.17) *satisfies the condition* (3.8).

Proof. For any node $\alpha \in N_i$, the following is satisfied by the definitions:

$$\mathcal{X}_{P_{\mathrm{DC}}[\alpha]} = \{X \in 2^{E^{\geq i}} \mid \forall v \in V[E^{\geq i}], \phi_v(\alpha) + deg_{G[X]}(v) \in dc(v)\}. \tag{3.18}$$

For any pair of nodes $\beta, \beta' \in N_i$, because (3.18) depends on only the configuration (3.17), we have

$$\phi(\beta) = \phi(\beta') \Rightarrow \mathcal{X}_{P_{\mathrm{DC}}[\beta]} = \mathcal{X}_{P_{\mathrm{DC}}[\beta']}. \tag{3.19}$$

□

generateNode Function

Initially, $\phi(\rho)$ has no information. Subsequently, for any node $\alpha \in N_i$ and $b \in \{0,1\}$, generateNode function computes $\phi(\alpha_b)$ as follows:

1. The function copies $\phi(\alpha_b)$ from $\phi(\alpha)$.
2. The function set 0 to $\phi_v(\alpha_b)$ for all $v \in F_{i+1} \setminus F_i$.
3. For all $v \in e_i$, the function updates $\phi_v(\alpha_b)$ by $\phi_v(\alpha_b) + b$.
4. The function deletes $\phi_v(\alpha_b)$ for all $v \in F_i \setminus F_{i+1}$.

$\perp$-prune Function

For any node $\alpha \in N_i$, the condition $\mathcal{X}_{P_{\mathrm{DC}}[\alpha_b]} = \emptyset$ can be detected by checking the following condition: the degree of a vertex $v \in e_i$ cannot belong $dc(v)$ finally. Note that missed cases can occur. Let $[j:k] := [j, j+k]$ for short. We can design the $\perp$-prune function as follows:

$$\perp\text{-prune}(\alpha, e_i, b) := \begin{cases} \mathit{True} & (\exists v \in e_i, [\phi_v(\alpha) + b : deg_{G[E^{\geq i+1}]}(v)] \cap dc(v) = \emptyset), \\ \mathit{False} & (\text{otherwise}). \end{cases} \tag{3.20}$$

Fortunately, the $\perp$-prune function leads the assumption in Lemma 3.3.2.

$\top$-prune Function

It is enough that the $\top$-prune function detects the condition of $\mathcal{X}_{P_{\mathrm{DC}}[\alpha_b]} = \{\emptyset\}$ at last step. Thus, We can easily design the $\top$-prune function for the BDD/ZDD as follows:

$$\top\text{-prune}(\alpha, e_i, b) := \begin{cases} \mathit{True} & (i = n), \\ \mathit{False} & (\text{otherwise}). \end{cases} \tag{3.21}$$

Note that the $\bot$-prune function is evaluated before the $\top$-prune function and can exactly detect the condition $\mathcal{X}_{P_{\mathrm{DC}}[\alpha_b]} = \emptyset$ at last step. Then, we can solve the SEP P_{DC} by the TD-DD (a variant of the FBS).

3.4 Time Complexity

Let N^* be an upper bound of the number of generated nodes in each step: $|N_i| \leq N^*$ for any $i \in [n]$. Let λ be the computation time of each node, namely, the total computation time of generateNode function, $\bot$-prune function, and $\top$-prune function. Then, the TD-DD takes $O(\sum_{i\in[n]} \lambda |N_i|) = O(n\lambda N^*)$ time.

Because the number of generated nodes is equal to the number of distinct configurations, we should estimate it. For example, N^* of the TD-DD for P_{KP} is equal to the given capacity W. In addition, λ is constant, because we can easily precompute $w(S^{\geq i+1})$ and $w^*(S^{\geq i})$. Then, the TD-DD for P_{KP} takes $O(nW)$ time which is equal to that of the basic dynamic programming for the KP on integer space.

3.5 Concluding Remarks for TD-DD

In this chapter, we presented a generalized framework TD-DD for a top-down construction of decision diagram to solve SEPs efficiently. In addition, we introduced two examples of TD-DD itemsets with knapsack-type constraints and degree constrained subgraphs. Subsequently, we discussed about the time complexity and analyzed the TD-DD for knapsack-type constraints. In the following chapters, we solve some SEPs and various combinatorial problems by using the TD-DD.

Chapter 4

Implicit Enumeration of Feasible Mine Assignments on Minesweeper Puzzles

Puzzles are general entertainments and topics of combinatorial problems for a long time: for example the Sudoku [41, 43], the N-queens problem [8, 14], and link puzzles [68]. The Minesweeper puzzle is also a popular puzzle game, which is frequently bundled with operating systems and GUIs such as Windows, X11, KDE, and GNOME. A Minesweeper puzzle is played on a grid board with closed cells. The goal of the puzzle is to open all safety cells avoiding mine cells while referring hints: Safety cells has a hint number of hidden mines in adjacent cells.

The Minesweeper puzzle has been formalized as some combinatorial problems and studied as a topic of the computational complexity theory: The *Minesweeper consistency problem* (MCP) [36] is a decision problem to decide whether a given fixed Minesweeper board has a feasible mine assignment or not. The MCP belongs to the class of NP-complete [36] even if the board is one dimensional [19]. In contrast, the *Minesweeper counting problem* (#M) [49] requires counting the number of feasible mine assignments. The *Minesweeper constrained counting problem* (#MC) [22] append a constraint, which is the total number of mines, to #M. The #M and the #MC belong to the class of #P-complete. Moreover, because the Minesweeper board can be represented by graphs, the problems above are also studied on general graph-based boards. However, any study on the Minesweeper puzzle is not interested in specific feasible mine assignments. This is a major difference as compared with studies on other puzzles such as N-queens problem.

In this chapter, we define the *Minesweeper Generation Problem* (MGP) which is a new problem on the Minesweeper puzzle: The MGP is an SEP which requires obtaining all feasible mine assignments on a given fixed Minesweeper board. An interesting thing is that the MGP includes the problems MCP and #M, i.e., enumeration solves the decision problem and the counting problem. Therefore, MGP is harder than or equal to

both of the MCP and the #M.

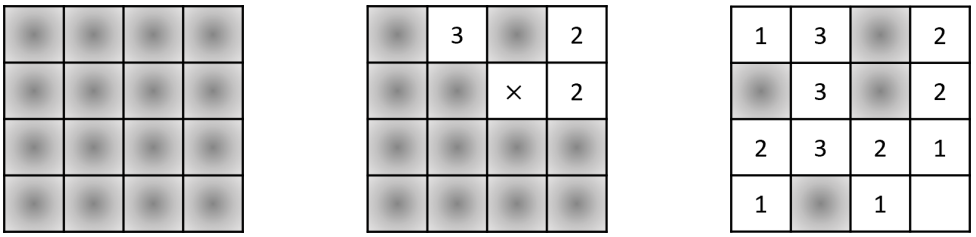
Our main contribution is to propose two approaches for solving MGPs by ZDDs and compare their performances: One is a naive combinatorial approach that uses a bottom-up algorithm combined with the family algebra. The other is a one-pass approach using the TD-DD with the concept of degree constrained subgraphs. Subsequently, we show that our approaches can be applied to the MCP, the #M, and the #MC. Moreover, we consider a situation to play a Minesweeper puzzle with a simple strategy such that guessing the mine probability of each cell. Using our approaches, we can compute the exact mine probability. Finally, we show experimental results to evaluate the performance of our approaches that indicates the approach using the TD-DD is better on various instances.

4.1 Preliminaries

In this section, we introduce the formal rules of the popular Minesweeper puzzle. Subsequently, we define generalized minesweeper board and the Minesweeper generation problem which is treated in this chapter.

4.1.1 Minesweeper Puzzles

The popular Minesweeper puzzle is played on a grid board initialized by closed cells such as Figure 4.1a. Each cell hides a mine, a hint number, or nothing. Here, we consider that cells with nothing has a hint number zero. The player opens a closed cell in each step. If the opened cell hides a mine, the player will lose (Figure 4.1b). Otherwise, the player obtains a hint: the hint number k means that there are exactly k hidden mines in the adjacent cells of the eight neighbors. The player will win when all the cells with a hint number or nothing are opened (Figure 4.1c).



(a) An initial grid board (b) A lose situation (c) A win situation

Figure 4.1. A popular Minesweeper puzzle

4.1.2 Minesweeper Generation Problem

Because the Minesweeper puzzle uses adjacent relations between cells with a hint number and closed cells, we can define a fixed Minesweeper board by $M := (G, h)$ with a *bipartite* graph $G = (U \cup V, E)$ and a function $h\colon U \to \mathbb{N}$:

- Each vertex $u \in U$ represents a opened cell with the hint number $h(u)$.
- Each vertex $v \in V$ represents a closed cell.
- Each edge $\{u, v\} \in E$ represents an adjacent relation between cells represented by $u \in U$ and $v \in V$.

For each vertex $u \in U$, let $A(u)$ be the set of all the adjacent vertices representing a closed cell: $A(u) := \{v \in V \mid \{u, v\} \in E\}$. Note that we ignore the cells with nothing and the unopened cells with no adjacent hint in short. For example, the bipartite graph for a fixed 4×4 grid board is shown in Figure 4.2. This definition is also a generalization of fixed Minesweeper boards: We are also interested in non-grid boards represented by general bipartite graphs such as Figure 4.3.

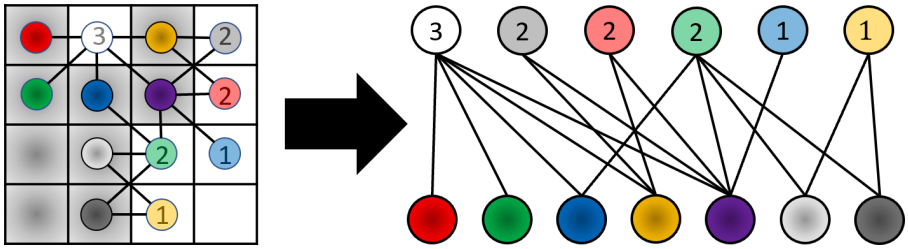


Figure 4.2. A 4×4 grid board and its graph representation

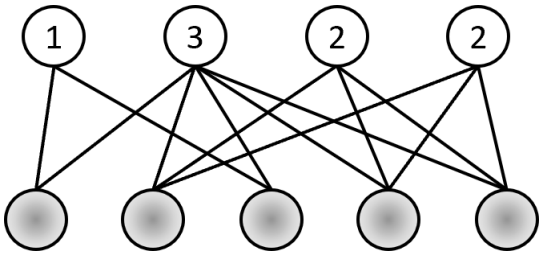


Figure 4.3. A graph for non-grid board. Namely, this graph cannot be embedded in grid boards.

We define the *Minesweeper generation problem* (MGP) on a fixed Minesweeper board M, which requires all the feasible mine assignments on M, as an SEP $P_{\mathrm{MS}} = \langle V, C_{\mathrm{MS}} \rangle$. The property function $C_{\mathrm{MS}}\colon 2^V \to \{0, 1\}$ is defined as

$$C_{\mathrm{MS}}(X) = 1 \iff \forall u \in U, |A(u) \cap X| = h(u). \tag{4.1}$$

Namely, $X \in 2^V$ has the property C_{MS} iff X represents a feasible mine assignment: We assign the mines to all the cells represented by X. For example, Figure 4.4 represents the solution for the MGP on the graph in Figure 4.3. In this chapter, we aim to solve MGPs efficiently by two approaches.

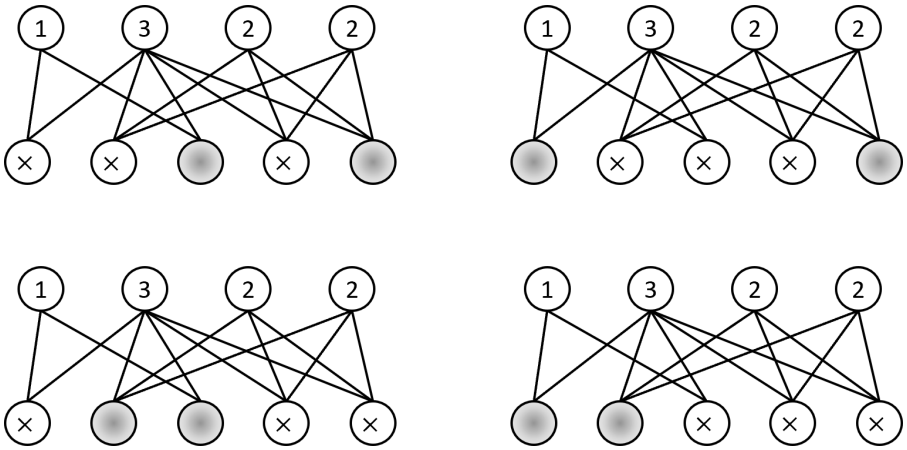


Figure 4.4. Solution for the MGP on the graph in Figure 4.3

4.2 Naive Combinatorial Approach Using Family Algebra

First, we present a naive combinatorial approach to solve MGPs by using the family algebra. Given a MGP P_{MS}, the output of the algorithm is a ZDD for $\mathcal{X}_{P_{\mathrm{MS}}}$.

The main idea is to divide P_{MS} as follows: For each vertex $u \in U$, let $P^u_{\mathrm{MS}} = \langle V, C^u_{\mathrm{MS}} \rangle$ be a sub-SEP with respect to u. The property function $C^u_{\mathrm{MS}} \colon 2^V \to \{0,1\}$ is defined as

$$C^u_{\mathrm{MS}}(X) = 1 \iff |A(u) \cap X| = h(u). \tag{4.2}$$

Then, we have

$$\mathcal{X}_{P_{\mathrm{MS}}} = \bigcap_{u \in U} \mathcal{X}_{P^u_{\mathrm{MS}}}. \tag{4.3}$$

Thus, we should construct the ZDDs for P^u_{MS} ($u \in U$) and conduct the intersection operations of them. For constructing objective ZDDs, we introduce a basic bottom-up algorithm in the following. Let $V := \{v_1, \ldots, v_n\}$ and $V^{\geq i} := \{v_i, \ldots, v_n\}$.

For each $u \in U$, let $\mathcal{Z}^u_{i,k}$ be a ZDD representing a set family $\mathcal{X}^u_{i,k} \subseteq^{V^{\geq i}}$ defined as

$$\mathcal{X}^u_{i,k} := \{X \in 2^{V^{\geq i}} \mid A(u) \cap X = k\}. \tag{4.4}$$

Then, we immediately have

$$v_i \in A(u) \Longrightarrow \mathcal{X}^u_{i,k} = \mathcal{X}^u_{i+1,k} \cup \left(\mathcal{X}^u_{i+1,k-1} \bowtie v_i \right), \tag{4.5}$$

and

$$v_i \notin A(u) \Longrightarrow \mathcal{X}^u_{i,k} = \mathcal{X}^u_{i+1,k}. \tag{4.6}$$

Therefore, we can construct $\mathcal{Z}^u_{i,k}$ as follows: Generate a node with label i where its 0-arc (resp. 1-arc) points the root node of $\mathcal{Z}^u_{i+1,k}$ (resp. the root node of $\mathcal{Z}^u_{i+1,k-1}$ if $v_i \in A(u)$, otherwise $\mathcal{Z}^u_{i+1,k}$). Then, the generated node becomes the root node of $\mathcal{Z}^u_{i,k}$. The algorithm conducts this process with the bottom-up manner from the terminals where $\bot = \mathcal{Z}^u_{n+1,-1}$ and $\top = \mathcal{Z}^u_{n+1,0}$ for convenience. Finally, we obtain $\mathcal{Z}^u_{1,h(u)}$ that is an objective ZDD in the computation time of $O(h(u)n)$, because node generations occur $O(h(u))$ times for each $i \in [n]$. The algorithm is shown in Algorithm 4.2.1. Finally, we conduct the intersection operations over $\mathcal{Z}^u_{1,h(u)}$ for all $u \in U$ to solve the P_{MS}.

Algorithm 4.2.1 A bottom-up construction of $\mathcal{Z}^u_{1,h(u)}$

```
for i = n, ..., 1 do
  for k = h(u), ..., 0 do
    Generate a node α as the root node of Z^u_{i,k}
    Set α_0 to the root node of Z^u_{i+1,k}
    if v_i ∈ A(u) then
      Set α_1 to the root node of Z^u_{i+1,k-1}
    else
      Set α_1 to the root node of Z^u_{i+1,k}
    end if
  end for
end for
return Z^{≥1}_{u,h(u)}
```

4.3 Graph-Based Approach Using Degree Constraints

Next, we present an approach to solve MGPs by using the concept of the degree constrained subgraph. Given an MGP P_{MS}, the approach converts P_{MS} into an SEP $P_{\mathrm{DC}} = \langle E, C_{\mathrm{DC}} \rangle$ with a degree constraint on the bipartite graph G. The output of the approach is a ZDD for $\mathcal{X}_{P_{\mathrm{DC}}}$ that has one-to-one correspondence with $\mathcal{X}_{P_{\mathrm{MS}}}$.

We define the degree constraint dc as

$$dc(p) = \begin{cases} \{h(p)\} & (p \in U), \\ \{0, deg_G(p)\} & (p \in V). \end{cases} \tag{4.7}$$

As with (3.16), we define the property function $C_{\mathrm{DC}} \colon 2^E \to \{0,1\}$ as

$$C_{\mathrm{DC}}(X) = 1 \iff \forall p \in (U \cup V), deg_{G[X]}(p) \in dc(p). \tag{4.8}$$

For converting each $X \in \mathcal{X}_{P_{\mathrm{DC}}}$ into a solution of the original MGP P_{MS}, we define the following function $f_M \colon \mathcal{X}_{P_{\mathrm{DC}}} \to \mathcal{X}_{P_{\mathrm{MS}}}$:

$$f_M(X) = \{v \in V \mid deg_{G[X]}(v) = deg_G(v)\} \tag{4.9}$$

We have the following lemma.

Lemma 4.3.1. *f_M is a bijection.*

Proof. For any $Y \in \mathcal{X}_{P_{\mathrm{MS}}}$, we select a subset $X := \{\{u,v\} \in E \mid u \in U, v \in Y\}$. Then, $deg_{G[X]}(u) = A(u) \cap Y = h(u)$ for all $u \in U$, $deg_{G[X]}(v) = deg_G(v)$ for all $v \in Y$, and $deg_{G[X]}(v) = 0$ for all $v \in V \setminus Y$. Hence, $X \in \mathcal{X}_{P_{\mathrm{DC}}}$ and $f_M(X) = Y$.

The degree constraint $dc(p) = \{0, deg_G(p)\}$ for all $p \in V$ means that any edge connecting p is not used or all edges connecting p are used. Note that any edge in E connects a vertex in U to a vertex in V. Then, for any two subsets $X, X' \in \mathcal{X}_{P_{\mathrm{DC}}}$, we have $f_M(X) = f_M(X') \Longrightarrow X = X'$. Thus, lemma follows. □

Note that P_{DC} is solved by the TD-DD according to Section 3.3.2. Therefore, lemma 4.3.1 says that the SEP P_{MS} is solved via the another SEP P_{DC}.

4.4 Applications

In this section, we show that our approach can be applied to various combinatorial problems related to the Minesweeper puzzle: the MCP, the #M, the #MC, and help with playing Minesweeper puzzles.

4.4.1 Solving Various Problems on Minesweeper Puzzles

The MCP, the #M, and the #MC can be defined by using the notations of the SEP P_{MS} as follows:

- Given a fixed Minesweeper board M, the MCP requires to answer whether M has a feasible mine assignment or not, i.e., $\mathcal{X}_{P_{\mathrm{MS}}} \neq \emptyset$ or $\mathcal{X}_{P_{\mathrm{MS}}} = \emptyset$.

- Given a fixed Minesweeper board M, the #M requires the number of the feasible mine assignments on M, i.e., $|\mathcal{X}_{P_{\mathrm{MS}}}|$
- Given a fixed Minesweeper board M and an integer $k \in \mathbb{N}$, the #MC requires the number of the feasible mine assignments on M with exactly k mines, i.e., $|\{X \in \mathcal{X}_{P_{\mathrm{MS}}} \mid |X| = k\}|$

If we have the solution of P_{MS}, we can easily solve the MCP and the #M by the definition: After obtaining a ZDD for $\mathcal{X}_{P_{\mathrm{MS}}}$ or $\mathcal{X}_{P_{\mathrm{DC}}}$, we use the Algorithm 2.4.2 to count the number of solutions.

For solving #MC, we construct a ZDD for the set family $\mathcal{X}_k$ where

$$\mathcal{X}_k := \{X \in 2^V \mid |X| = k\}. \tag{4.10}$$

Then, because $|\mathcal{X}_{P_{\mathrm{MS}}} \cap \mathcal{X}_k|$ is the solution of the #MC, we conduct the intersection operation between the ZDDs. A ZDD for $\mathcal{X}_k$ can be constructed by the TD-DD for a special case of knapsack-type constraints: We set the weight as $w(v) = 1$ for all $v \in V$ and the capacity as $W = k$.

Similarly, we utilize $\mathcal{X}_{P_{\mathrm{DC}}}$ for solving the #MC. Let R be a set of edges such that, for all $v \in V$, R has exactly one edge $e \in E$ where $v \in e$. We construct a ZDD for the set family $\hat{\mathcal{X}}_k$ where

$$\hat{\mathcal{X}}_k := \{X \in 2^E \mid |X \cap R| = k\}. \tag{4.11}$$

Then, because $|\mathcal{X}_{P_{\mathrm{DC}}} \cap \hat{\mathcal{X}}_k|$ is the solution of the #MC, we conduct the intersection operation between the ZDDs. A ZDD for $\hat{\mathcal{X}}_k$ can also be constructed by the TD-DD for a special case of knapsack-type constraints: We set the weight as $w(e) = 1$ for all $e \in R$ and $w(e') = 0$ for all $e' \in E \setminus R$ and the capacity as $W = k$.

4.4.2 Help with Playing Minesweeper Puzzles

On playing a Minesweeper puzzle, a simple strategy is to guess the *mine probability* for each closed cell, which is the probability that a mine is hidden in the cell. Our approach can help this strategy.

We formally define the mine probability $p(M; v)$ of vertex $v \in V$ on a fixed Minesweeper board M as follows:

$$p(M; v) := \frac{|\{X \in \mathcal{X}_{P_{\mathrm{MS}}} \mid v \in X\}|}{|\mathcal{X}_{P_{\mathrm{MS}}}|}. \tag{4.12}$$

Note that $\{X \in \mathcal{X}_{P_{\mathrm{MS}}} \mid v \in X\}$ can be replaced with $\{X \in \mathcal{X}_{P_{\mathrm{DC}}} \mid \exists e \in X, v \in e\}$. By using the family algebra, we have $\{X \in \mathcal{X}_{P_{\mathrm{MS}}} \mid v \in X\} = \mathcal{X}_{P_{\mathrm{MS}}} \lhd v$. Similarly, we have $\{X \in \mathcal{X}_{P_{\mathrm{DC}}} \mid \exists e \in X, v \in e\} = \mathcal{X}_{P_{\mathrm{DC}}} \lhd e'$ where $v \in e'$ and $e' \in E$. Thus, we can easily compute the mine probability by using the family algebra and the Algorithm 2.4.2.

4.5 Experiments

We conduct the experiments for evaluating the performance of two proposed algorithms. Especially, we use not only grid-based board but also graph-based board in the experiments. All the code was implemented in C++ (g++5.4.0 with the -O3 option). We used 64-bit Ubuntu 16.04 LTS with an Intel Core i7-3930K 3.2 GHz CPU and 64 GB RAM. The experiments were done on the PC with Intel Core i7-3930K 3.2GHz CPU and 64GB memory.

The instance boards were randomly generated as follows: Some of them are 30×30 grid, and the others are random graph with 300 vertices and 900 edges (relatively sparse). We set three types of probability that each cell hides a mine, 10%, 20%, and 30%. We also set nine types of probability that each cell has a hint number, 10%, 20%, . . . , and 90%.

Tables 4.1 and 4.3 summarize the computation time. 'NA' indicates the algorithm with a naive combinatorial approach, and 'DC' indicates the algorithm with degree constraints approach. Tables 4.2 and 4.4 summarize the number of feasible mine assignments in each instance. The best time is written in bold letters.

On the grid-based boards, the number of feasible mine assignments is over 10^{20} in quite of half instances. However, ' DC' shows the best time in most instances. Especially, it is 100 times faster than ' NA' n some instances. Thus, the degree constraints approach is efficient for the MGP with grid-based boards. But in instance consisting of 30% hidden mine and 40% hints number, 'DC' is inefficient in comparison with 'NA'. We consider that the reason depends on the complexity of the graph generated by the board; it is supposed that the degree constraints are easily complicated by connections of the vertices.

Table 4.1. Computation time (in second) for grid-based board

mine ratio	10%		20%		30%	
hint ratio	NA	DC	NA	DC	NA	DC
10%	3.602	**0.006**	2.917	**0.006**	2.872	**0.005**
20%	15.306	**0.023**	16.124	**0.095**	16.921	**0.024**
30%	19.812	**0.149**	23.974	**0.732**	22.268	**3.472**
40%	24.636	**0.051**	34.478	**1.226**	**36.521**	79.872
50%	23.334	**0.041**	33.440	**0.331**	37.740	**1.884**
60%	18.915	**0.033**	23.872	**0.065**	35.572	**0.273**
70%	11.826	**0.028**	17.801	**0.031**	23.063	**0.035**
80%	6.013	**0.019**	12.659	**0.024**	18.790	**0.030**
90%	1.788	**0.013**	6.128	**0.018**	11.642	**0.023**

Table 4.2. Number of feasible mine assignments for grid-based board

hint ratio\mine ratio	10%	20%	30%
10%	1.53×10^{30}	6.33×10^{47}	1.92×10^{55}
20%	3.76×10^{30}	1.35×10^{55}	8.41×10^{71}
30%	1.12×10^{20}	8.29×10^{37}	2.81×10^{51}
40%	4.09×10^{7}	5.24×10^{21}	5.90×10^{39}
50%	512	1.73×10^{7}	6.89×10^{25}
60%	16	65536	3.52×10^{6}
70%	1	64	6912
80%	1	2	4
90%	2	1	1

On the graph-based board, ‘DC’ shows the best time in all instances. Especially, it is 100 times faster than ‘NA’ in some instances. The computation time of ‘NA’ is not stable compared with grid-based boards. On the other hand, the computation time of ‘dc’ is stable. We consider that the result is caused by dispersion of the degree in the original graph, which affect number of adjacent cells. In contrast, number of adjacent cells of each cell in grid is approximately constant. Thus, the degree constraints approach is also efficient for the MGP with boards based on sparse graphs.

Table 4.3. Computation time (in second) for graph-based board ($|V| = 300, |E| = 900$)

mine ratio	10%		20%		30%	
hint ratio	NA	DC	NA	DC	NA	DC
10%	0.171	**0.001**	0.629	**0.002**	52.186	**0.001**
20%	0.816	**0.004**	89.602	**0.024**	82.493	**0.141**
30%	1.156	**0.008**	100.224	**0.025**	49.926	**0.650**
40%	5.859	**0.012**	12.410	**0.071**	60.105	**2.592**
50%	1.189	**0.008**	81.190	**0.007**	65.981	**0.017**
60%	0.917	**0.006**	2.065	**0.007**	147.352	**0.234**
70%	0.461	**0.006**	0.799	**0.008**	2.206	**0.008**
80%	0.203	**0.005**	0.500	**0.008**	0.737	**0.007**
90%	0.051	**0.003**	0.184	**0.005**	0.398	**0.005**

Table 4.4. Number of feasible mine assignments for graph-based board

hint ratio\mine ratio	10%	20%	30%
10%	10251360	3.63×10^9	1.23×10^{15}
20%	2419200	7.65×10^{10}	1.75×10^{14}
30%	4	7.97×10^7	7.15×10^{10}
40%	90	11520	1.28×10^8
50%	4	12	864
60%	1	4	4
70%	1	1	2
80%	1	1	1
90%	1	1	1

4.6 Concluding Remarks for Minesweeper Puzzles

In this chapter, we develop the MGP which is a problem to generate all feasible mine assignments on a given fixed Minesweeper board. We also proposed two algorithms for implicit enumeration by ZDDs to solve the MGP. One is a naive combinatorial approach combined with the family algebra. The other is a degree constraints approach using a formulation by degree constrained subgraph and the TD-DD. Subsequently, we showed their applications to the various problems related to the Minesweeper puzzle such as the MCP, #M, #MC, and the computation of mine probability. An important contribution was that we compared performances of the approaches on various instances: the experimental results showed that the approach using the TD-DD was better than the naive combinatorial approach.

As a future work, we can consider an online problem for the Minesweeper puzzle, where hints are given one by one. Solving the online problem is close to actual play of the Minesweeper puzzle.

Chapter 5

Implicit Enumeration of Strongly Connected Spanning Subgraphs on Directed Graphs

On network systems, the connections are a fundamental component, such as communication between facilities. A network system is often represented by a graph; the vertices represent facilities such as servers, and the edges represent connections between facilities such as cables. On undirected graphs, the property called *spanning* is commonly used for representing the condition that all the vertices are connected, i.e, they can communicate with each other. As a natural extension of spanning, the directed version can be considered: A directed graph is said to be *strongly connected* if any pair of vertices has bidirectional paths. On the directed graph, strongly connected leads the condition that all the vertices can communicate with each other. Because strongly connected is demanded in various network systems such as ad-hoc network, the property has been applied to various real-world issues such as evaluation of *network reliability* [13, 23, 32, 50, 66] and visualization of network systems [1, 2, 4, 65].

On modeling the issues above, subgraphs called *strongly connected spanning subgraphs* (SCSSs) have frequently been used. A subgraph is an SCSS if it is strongly connected and has all the vertices of an original graph. Network reliability is the probability that a network system can perform a desired operation, such as communication between facilities, against stochastic equipment failures. Network reliability on directed graphs is called the *strongly connected reliability* (SCR) [13] which is the probability that the graph maintains strongly connected. On visualization of network systems, desirable graphs are removed complexity of connections as much as possible with maintaining global connections, namely, small and strongly connected. It is formalized as the *minimum SCSS problem* (min-SCSS) to obtain the SCSS with smallest number of edges. However, the exact computation of the SCR belongs to the class of #P-complete [63].

Similarly, the min-SCSS belongs to the class of NP-hard [20]. Therefore, efficiently dealing with SCSSs is an important issue.

In this chapter, we deal with an SEP to obtain all the SCSSs on a given directed graph. Especially, we propose the TD-DD (a new variant of the FBS) for constructing BDDs for SCSSs. Once a BDD for the SCSSs is obtained, we can efficiently compute the exact SCR and obtain the minimum SCSS by functions of the BDD. Particularly, because the BDD has several functions to search solutions under various conditions, it can be flexibly utilized. We conducted computational experiments to evaluate our algorithm. We used several real-world and synthetic networks with a few hundred edges. The experimental results showed that our algorithm can construct the BDDs for the SCSSs in a reasonable time. We also computed the SCR on each instance using the constructed BDDs, which was previously impossible.

5.1 Strongly Connected Spanning Subgraphs

Here, we introduce the definition and the notations of the strongly connected spanning subgraph (SCSS) on a directed graph $G = (V, E)$.

Definition 5.1.1. *For any pair of vertices $(u, v) \in V^2$, v is* reachable *from u on G, which is denoted by $u \rightsquigarrow_G v$, if G has a u-v path. Similarly, $u \not\rightsquigarrow_G v$ denotes that v is not reachable from u on G.*

Definition 5.1.2. *G is said to be* strongly connected *if $u \rightsquigarrow_G v$ for all $(u, v) \in V^2$.*

Definition 5.1.3. *For any edge subset $X \subseteq E$, $G[X]$ is said to be* spanning *if $V[X] = V$.*

Definition 5.1.4. *For any edge subset $X \subseteq E$, the edge induced subgraph $G[X]$ is an* SCSS *of G if $G[X]$ is strongly connected and spanning. If $G[X]$ is an SCSS, X is a* strongly connected edge subset (SCES) *of G.*

In this chapter, we treat the SEP $P_{\mathrm{SC}} = \langle E, C_{\mathrm{SC}} \rangle$ on G where the property function C_{SC} is defined as

$$C_{\mathrm{SC}}(X) = 1 \iff \forall (u, v) \in V^2, u \rightsquigarrow_{G[X]} v. \tag{5.1}$$

Namely, $\mathcal{X}_{P_{\mathrm{SC}}}$ is the set of all the SCESs of G. Note that we assume that the input graph G is strongly connected. Figure 5.1 shows an example of all the SCSSs of a strongly connected graph.

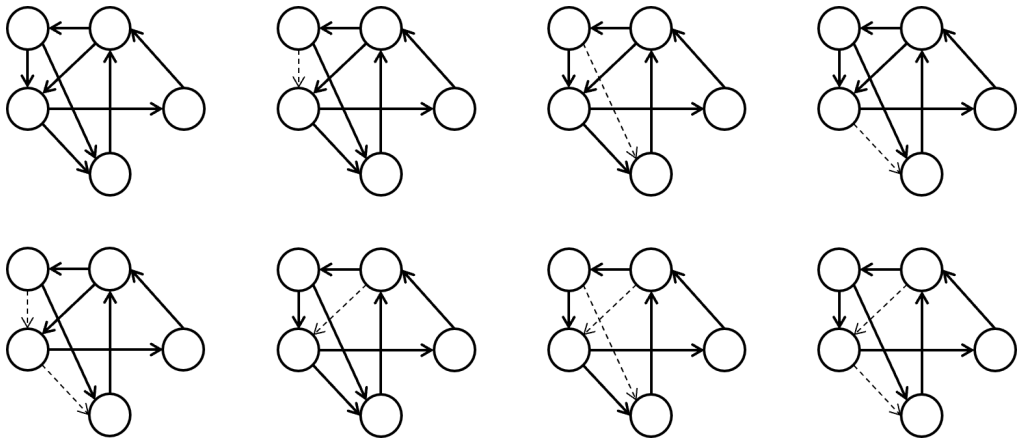


Figure 5.1. All the SCSSs of a graph. The left-top graph is the original and itself is an SCSS.

5.2 Proposed Method

We design four primary components to construct a BDD for $\mathcal{X}_{P_{\mathrm{SC}}}$ by a TD-DD. Subsequently, we analyze its time complexity. Let $E := \{e_1, \ldots, e_n\}$, $E^{<i} := \{e_1, \ldots, e_{i-1}\}$, and $E^{\geq i} := \{e_i, \ldots, e_n\}$. In the following, we assume that $e_i = (s_i, t_i)$ for each $i \in [n]$.

5.2.1 Top-Down Construction of BDDs for SCSSs

Configuration

For each $i \in [n]$, i-th frontier is defined as $F_i := V[E^{<i}] \cap V[E^{\geq i}]$ as with Section 2.4.5. We use the reachability on the frontier vertices as the configuration. For any node $\alpha \in N_i$, $\phi(\alpha)$ is defined as a reachability matrix indexed by F_i^2:

$$\phi(\alpha)_{u,v} := \begin{cases} 1 & (\forall X \in \bar{\mathcal{D}}(\alpha), u \rightsquigarrow_{G[X]} v), \\ 0 & (\text{otherwise}). \end{cases} \tag{5.2}$$

We assume that $\bot$-pruning is always conducted if possible. This assumption deduces that each vertex excluded from the past frontier is reachable from (resp. to) the new frontier on $G[X]$ ($X \in \bar{\mathcal{D}}(\alpha)$). By this assumption, we obtain the following lemma.

Lemma 5.2.1. *The configuraiton* (5.2) *satisfies the condition* (3.8).

Proof. For any node $\alpha \in N_i$, let E_α be a *contracted edge set* of G, which is derived from the configuration $\phi(\alpha)$, defined as

$$E_\alpha := \{(u,v) \in F_i^2 \mid \phi(\alpha)_{u,v} = 1\}. \tag{5.3}$$

Similarly, let G_α be a *contracted graph* of G defined as

$$G_\alpha := (V[E^{\geq i}], E^{\geq i} \cup E_\alpha). \tag{5.4}$$

According to the assumption above, each edge subset $X \in \mathcal{X}_{P_{\mathrm{SC}}[\alpha]}$ satisfies that $X \cup E_\alpha$ is an SCES of G_α. Therefore, we have

$$C_{\mathrm{SC}}[\alpha](X) = 1 \iff \forall (u,v) \in V[E^{\geq i}]^2, u \rightsquigarrow_{G_\alpha[X \cup E_\alpha]} v. \tag{5.5}$$

Let $\beta, \beta' \in N_i$. Because (5.5) depends on only the configuration (5.2), we have

$$\phi(\beta) = \phi(\beta') \Rightarrow \mathcal{X}_{P_{\mathrm{SC}}[\beta]} = \mathcal{X}_{P_{\mathrm{SC}}[\beta']}. \tag{5.6}$$

□

generateNode Function

Initially, $\phi(\rho)$ has no information. Subsequently, for any node $\alpha \in N_i$ and $b \in \{0,1\}$, generateNode function computes $\phi(\alpha_b)$ as follows:

1. Remove the rows and columns corresponding to the vertices excluded from the frontier, i.e, $F_i \setminus F_{i+1}$.
2. Inset the rows and columns corresponding to the vertices included in the frontier, i.e, $F_{i+1} \setminus F_i$.
3. If $b = 0$, end the update. Otherwise, including a new edge changes the reachability; thus, we update $\phi(\alpha_b)$ as the transitive closure of the frontier.

$\perp$-prune Function

Here, we design a $\perp$-prune function that does not contradict the assumption of Lemma 5.2.1. We use the following properties of $C_{\mathrm{SC}}[\alpha]$ where $\alpha \in N_i$:

- $C_{\mathrm{SC}}[\alpha](X) = 1$ implies $C_{\mathrm{SC}}[\alpha](X \cup \{e_i\}) = 1$ for any $X \subseteq E^{\geq i}$, because an edge subset that contains an SCES is also an SCES.
- If $\phi(\alpha)_{s_i,t_i} = 1$, then $C_{\mathrm{SC}}[\alpha](X) = 1$ implies $C_{\mathrm{SC}}[\alpha](X \setminus \{e_i\}) = 1$ for any $X \subseteq E^{\geq i}$, because $e_i = (s_i, t_i) \in E_\alpha$.

Thus, we have a chance to conduct $\perp$-pruning for the case that e_i is excluded and $\phi(\alpha)_{s_i,t_i} = 0$ is satisfied.

Let $G - e := (V, E \setminus \{e\})$ be a graph that is obtained by removing an edge e from G. A graph $G_\alpha - e_i$ has no SCES if and only if $\{e_i\}$ forms a cut set from s_i to t_i. Therefore, we design $\perp$-prune(α, e_i, b) as:

$$\perp\text{-prune}(\alpha, e_i, b) = \begin{cases} \mathit{True} & (b = 0,\ \phi(\alpha)_{s_i,t_i} = 0, \text{ and } s_i \not\rightsquigarrow_{G_\alpha - e_i} t_i), \\ \mathit{False} & (\text{otherwise}). \end{cases} \tag{5.7}$$

$\top$-prune Function

Similarly, we have a chance to conduct $\top$-pruning for the case that e_i is included. The following property is observed: $C_{\mathrm{SC}}[\alpha](\{e_i\}) = 1$ implies $C_{\mathrm{SC}}[\alpha](\{e_i\} \cup X) = 1$ for any $X \subseteq E^{\geq i+1}$. Therefore, we design $\top$-prune(α, e_i, b) as

$$\top\text{-prune}(\alpha, e_i, b) = \begin{cases} \mathit{True} & (\text{b=1}, \forall (u,v) \in V[E^{\geq i}]^2, u \rightsquigarrow_{G_\alpha[E_\alpha \cup \{e_i\}]} v), \\ \mathit{False} & (\text{otherwise}). \end{cases} \tag{5.8}$$

5.2.2 Time Complexity

For updating the configuration, the number of removed (resp. Insert) rows and columns is constant because $|F_i \setminus F_{i+1}|$ (resp. $|F_{i+1} \setminus F_i|$) is 0, 1, or 2. Updating $\phi(\beta)$ is performed in $O(|F_i|^2)$ time by the BFS/DFS on the frontier. Thus, our generateNode function can be processed in $O(|F_i|^2)$ time.

For evaluating $\bot$-prune(α, e_i, b) efficiently, we precompute the transitive closure of $G[E^{\geq i+1}]$. Using the transitive closure of $G[E^{\geq i+1}]$, the reachability from s_i to t_i on $G_\alpha - e_i$ is verified in $O(|F_i|^2)$ time by the BFS/DFS on the frontier. The precomputation of the transitive closures can be efficiently performed in decreasing order $i = n, \ldots, 1$; We compute the transitive closure of $G[E^{\geq i}]$ as the extension of the transitive closure of $G[E^{\geq i+1}]$ by the BFS/DFS on the $V[E^{\geq i}]$. Although the precomputation requires $O(\sum_{i=1}^{n} |V[E^{\geq i}]|^2)$ time, it is typically much faster than the FBS.

For evaluating $\top$-prune(α, e_i, b) efficiently, we precompute an integer

$$r := \min\{i \in \{1, \ldots, m\} \mid V[E^{\leq i}] = V\}. \tag{5.9}$$

Subsequently, $i < r$ implies $\top$-prune$(\alpha, e_i, 1) = \mathit{False}$ as $V[X \cup \{e_i\}] \neq V$ for any $X \in \bar{\mathcal{D}}(\alpha)$. If $i \geq r$, $\top$-prune$(\alpha, e_i, 1)$ is evaluated by the BFS/DFS on the frontier with computation time $O(|F_i|^2)$.

The time complexity of the proposed algorithm strongly depends on the number of possible configurations for each step $i \in [n]$, i.e., the number of nodes $|N_i|$ as mentioned in Section 3.4. Although we have $|N_i| \leq 2^{|F_i|^2}$, it is a rough estimation because configurations represent transitive closures. Let $T(k)$ be the number of transitive closures on k labeled vertices. It is known as OEIS A006905 (https://oeis.org/A006905) in the literature [51]. Then, we have $|N_i| \leq T(|F_i|)$. We show a partial table of $T(k)$ and 2^{k^2} in Table 5.1.

As a result, because each node $\alpha \in N_i$ takes $O(|F_i|^2)$ time as mentioned above, the total time complexity of the algorithm is $O\left(\sum_{i=1}^{n} |V[E^{\geq i}]|^2 + \sum_{i \in [n]} |F_i|^2 T(|F_i|)\right) = O(EV^2 + E\hat{F}^2 T(\hat{F}))$ where $\hat{F} = \max_{i \in [n]} |F_i|$. Although $T(k)$ is still huge as in Table 5.1, we show that the algorithm run on graphs of moderate sizes in the experimental results below.

Table 5.1. A partial table of $T(k)$ and 2^{k^2} $(1 \leq k \leq 10)$

k	$T(k)$	2^{k^2}
1	2	2
2	13	16
3	171	512
4	3994	65536
5	154303	33554432
6	9415189	68719476736
7	878222530	562949953421312
8	122207703623	1844674407370955161 6
9	24890747921947	241785163922925834941 2352
10	7307450299510288	126765060022822940149670320 5376

5.3 Applications

In the previous section, we present a new variant of the FBS that constructs a BDD for $\mathcal{X}_{P_{\mathrm{SC}}}$. The constructed BDD also allows us to solve SCSS-related problems efficiently.

5.3.1 Exact Computation of Strongly Connected Reliability

The strongly connected reliability (SCR) of G is the probability that G remains strongly connected after stochastic edge dropping. Let $\sigma(G)$ be the SCR of G. We assume that each edge e_i independently drops with probability $p(e_i)$. Then $\sigma(G)$ is defined as:

$$\sigma(G) := \sum_{X \in \mathcal{X}_{P_{\mathrm{SC}}}} p(X) \tag{5.10}$$

where

$$p(X) := \prod_{e_i \in X} (1 - p(e_i)) \prod_{e_j \in E \setminus X} p(e_j). \tag{5.11}$$

Namely, the SCR is equal to the occurrence probability of SCESs. Thus, once a BDD $\mathcal{B}$ for $\mathcal{X}_{P_{\mathrm{SC}}}$ is obtained, we can compute the SCR $\sigma(G)$ by the Algorithm 2.4.3 in the computation time $O(|\mathcal{B}|)$.

5.3.2 Finding Minimum SCSS

Given a weight function $w\colon E \to \mathbb{N}$, let $w(X) := \sum_{e \in X} w(e)$. An SCSS that has the smallest total weight of edges is called the *minimum* SCSS. The SCES of the minimum

SCSS is defined as follows:

$$X^* \in \arg\min_{X \in \mathcal{X}_{P_{\mathrm{SC}}}} w(X). \tag{5.12}$$

This is a linear optimization (minimization). Thus, once a BDD $\mathcal{B}$ for $\mathcal{X}_{P_{\mathrm{SC}}}$ is obtained, we can obtain X^* by the Algorithm 2.4.5 and the Algorithm 2.4.7 in the computation time $O(|\mathcal{B}|)$.

5.4 Experiments

We conducted computational experiments to evaluate the proposed algorithm. All the codes were implemented in C++ (g++4.8.4 with the -O3 option) using the TdZdd library [29], which is a highly optimized implementation for the FBS framework. All experiments were conducted on a 64-bit Ubuntu 16.04 LTS with an Intel Core i7-3930K 3.2 GHz CPU and 64 GB RAM.

5.4.1 Scalability on Synthetic Networks

First, to observe the performance of the proposed algorithm, we applied our method to two classes of synthetic networks. The first class was $5 \times w$ gird graphs that had $5w$ vertices, $18w - 10$ directed edges (undirected edges were replaced with two directed edges in both directions), and a pathwidth of 5. The second class was random graphs that had the same number of vertices of $5 \times w$ grid, $9w - 5$ directed edges, and a pathwidth of $\Theta(n)$. We used the algorithm proposed in [45] to generate the strongly connected random graphs. For each $w \in \{5, \ldots, 20\}$, one hundred random graphs were generated, and we evaluated the average performance on the random graphs.

The results of the synthetic networks are shown in Figure 5.2 and 5.3. The grid graphs show that the computation time increased slowly; our algorithm executed in 16 seconds for $n = 100$. However, for the random networks, the computation time increased rapidly for networks with $80 \leq |V|$ vertices. These results show that the large pathwidth affected the computation time of our algorithm.

The size of the constructed BDDs had a similar tendency with the computation times. The BDD size was increased slowly for the grid graphs and rapidly for the random networks with $80 \leq |V|$ vertices. Meanwhile, the sizes of the reduced BDDs were sufficiently small.

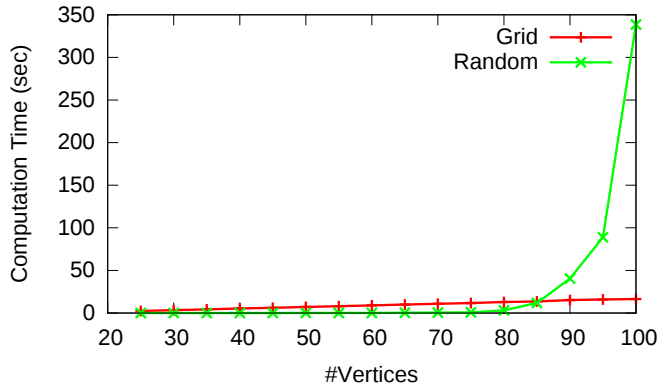


Figure 5.2. Computational time on $5 \times w$ grid graphs and random graphs.

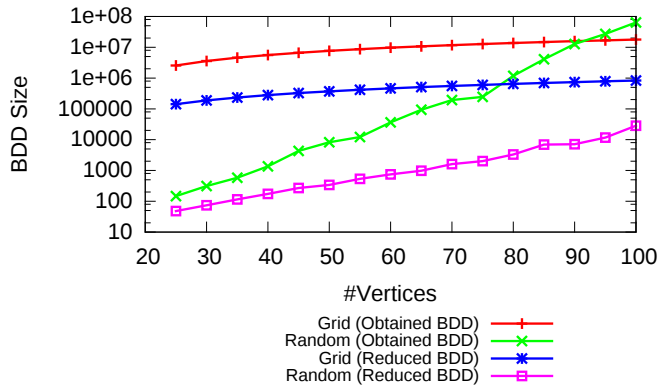


Figure 5.3. BDD size on $5 \times w$ grid graphs and random graphs.

5.4.2 Scalability on Real-World Networks

Next, to evaluate the practical performance of the proposed algorithm, we applied our method to real-world networks. The real-world graphs were obtained from SNDlib [70]. All the self-loops and multiple edges are deleted. Because all the graphs were undirected, we replaced each edge with two directed edges in both directions.

The results are shown in Table 5.2. The algorithm succeeded in constructing the BDDs for $\mathcal{S}_G$ on almost all the networks, however failed on three networks due to the memory limit. Although each succeeded instance might have a few hundred edges, the algorithm executed in a few seconds or a few minutes. By comparing a brain network and the failed ones, we found that the computational cost of the algorithm depends on the network structure.

As shown in the cardinality column, the numbers of SCESs are enormous. This implies that the naive exhaustive search is unrealistic. Particularly, the ta2 network has

$$2,320,225,475,355,945,207,334,621,674,664,990,580,848,170,679,757,701,120$$

($\simeq 2.3 \times 10^{54}$) SCESs. This shows the advantage of our approach; the BDD representation of the SCESs is efficient.

5.4.3 Computation of SCRs

We also conducted the experiments to compute the exact SCR using the constructed BDDs for analyzing the reliability of each network used in the experiments above. Once the BDDs are obtained, we can easily compute the exact SRC iteratively for various settings of the edge dropping probability. Therefore, we used the probability of the edge dropping that was moved from 1.0 to 0.0 and decreased by 0.01. The results are shown in Figure 5.4.

For the synthetic networks, the grid graphs have relatively high reliability until the edge dropping probability is less than 0.1, whereas the reliability of the random graphs was much lower. We consider that it is attributable to the sparsity of the random graphs; the number of edges is $9w-5$ in contrast with the number of vertices $5w$.

The real-world networks brain, ta2, and zib54 have low reliability; meanwhile, the newyork and pdh networks have much higher reliability. By comparing these networks, each of the networks with high reliability tends to have relatively many edges in contrast with the number of vertices. Although this may be a natural consequence, its practical verification was impossible previously.

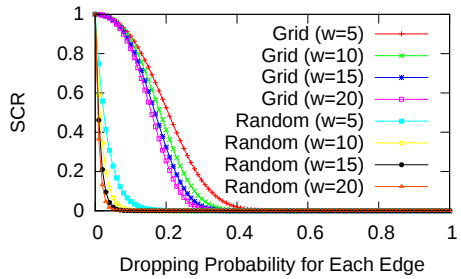
5.5 Discussion and Concluding Remarks for Strongly Connected Spanning Subgraphs

In this chapter, we proposed an algorithm to construct a BDD representing all SCSSs by the TD-DD, which is a new variant of the FBS. Moreover, we applied the algorithm to compute the exact SCR and obtain the minimum SCSS. The experimental results showed that the proposed algorithm ran on the real-world networks with a few hundred edges, even though they have a large number of SCSSs such as 2.3×10^{54}. The algorithm also succeeded to compute the SCR of the networks above, which was previously impossible.

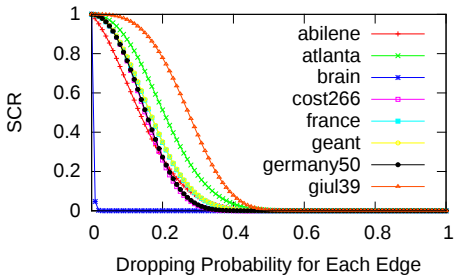
Our algorithm implicitly solved the enumeration problem of SCSSs. Regarding the explicit enumeration, only the algorithm to enumerate the *minimal* SCSSs is known [12]. Hence, we first tackled the enumeration problem of general SCSSs. Moreover, the enumeration algorithm in [12] requires the computation time depending on the number of minimal SCSSs, whereas our algorithm requires the computation time depending on the pathwidth of the given graph. This is an advantage of our algorithm in several graph classes having a small pathwidth and many minimal SCSSs.

Table 5.2. Computational results on real-world networks. Time denotes the time to construct the BDDs, BDD Size 1 denotes the size of the constructed BDDs, BDD Size 2 denotes the size of the reduced BDDs, and Cardinality denotes the number of SCESs. For the last three networks, the algorithm failed due to the memory limit.

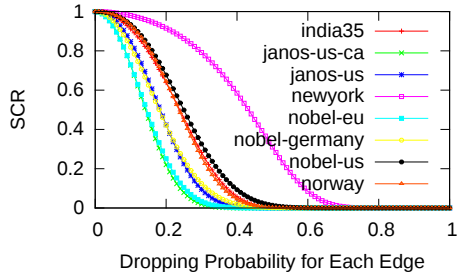
Network Name	$\|V\|$	$\|E\|$	Time (sec)	BDD Size 1	BDD Size 2	Cardinality
abilene	12	30	0.00	284	94	1.2e+06
atlanta	15	44	0.01	4,964	630	5.9e+10
brain	161	332	2.31	9,095	2,990	1.6e+07
cost266	37	114	0.46	382,680	22,221	5.3e+28
france	25	90	0.05	45,715	5,420	2.2e+23
geant	22	72	0.07	81,391	5,611	1.3e+18
germany50	50	176	138.56	124,052,168	3,454,355	1.3e+47
giul39	39	172	972.38	781,882,756	15,878,463	2.2e+49
india35	35	160	90.05	83,212,903	2,422,816	4.8e+45
janos-us-ca	39	122	0.37	416,769	29,749	7.4e+30
janos-us	26	84	0.08	59,486	5,320	3.4e+21
newyork	16	98	12.70	12,178,145	607,550	9.8e+28
nobel-eu	28	82	0.03	35,481	2,822	9.1e+19
nobel-germany	17	52	0.00	2,790	458	6.3e+12
nobel-us	14	42	0.01	15,526	2,285	6.1e+10
norway	27	102	0.28	325,280	19,543	1.2e+28
pdh	11	68	4.73	4,555,544	551,065	1.9e+20
pioro40	40	178	198.94	182,362,754	1,415,844	3.6e+51
polska	12	36	0.00	4,525	794	1.4e+09
sun	27	102	0.37	325,355	19,543	1.2e+28
ta1	24	102	0.08	52,991	4,489	2.5e+28
ta2	65	216	2.56	2,573,009	98,115	2.3e+54
zib54	54	160	0.18	158,979	13,571	1.2e+37
dfn-bwin	10	90	-	-	-	-
dfn-gwin	11	94	-	-	-	-
di-yuan	11	84	-	-	-	-



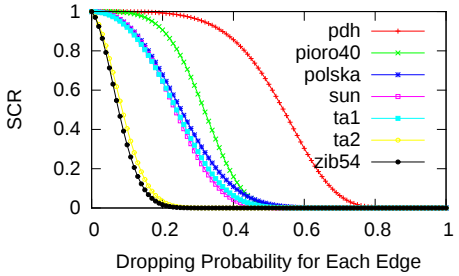
(a) Synthetic Networks
$w \in \{5, 10, 15, 20\}$



(b) Real-World Networks
(abilene $\sim$ giul39)



(c) Real-World Networks
(india35 $\sim$ norway)



(d) Real-World Networks
(pdh $\sim$ zib54)

Figure 5.4. The results of the SCRs computation.

An important future work is to compute (possibly approximate) SCRs in networks with large path-width. This may require new techniques such as the approximation of BDDs and the reduction of the networks.

Chapter 6

Implicit Enumeration of Pareto-Optimal Solutions for 0-1 Multi-Objective Knapsack Problems

Decision making with capacity is a frequent situation in the real-life, for example, resource allocation, selection of investments, and so on. They have often been formalized and solved as a combinatorial problem called the *0-1 knapsack problem* (KP): Given a set of items and a capacity, where each item has a value and a weight, the KP requires to find a subset having the maximum total value within the capacity. Many studies on the KP are shown in the literature [37]. However, this formulation is not always proper because values can include multiple criteria in the real situation. Therefore, the *0-1 multi-objective knapsack problem* (MKP), which is a *multi-objective combinatorial optimization* (MOCO) [44], has been studied. If each item has a value including m criteria, the MKP is said to be m-objective; The KP is the 1-objective case of the MKP. The MKP have many practical applications, for example, capital budgeting [53] and selection of transportation investment alternatives [59]. In the MOCO, desirable solutions have the property called the *Pareto-optimality*, i.e., a criterion cannot be improved without changing another criterion for the worse. If a solution has the Pareto-optimality, the solution called a *Pareto-optimal solution*. The goal of the MKP is to obtain a (possibly partial) set of Pareto-optimal solutions.

As a known result, although the KP belongs to the class of NP-hard, a pseudo-polynomial time algorithm by a dynamic programming (DP) runs on the KP efficiently. On the other hand, its simple extended algorithm does not efficiently run on the MKP because of increasing the dimension of the search space. Therefore, various approaches for the MKP have been studied: Metaheuristics such as simulated annealing and genetic algorithm efficiently find a set of approximate solutions, which are not necessarily a Pareto-optimal solution [21, 30, 64]. A labeling algorithm [18] and ε-constraint method

[16] find all Pareto-optimal solutions only for 2-objective cases. For general cases, the literature [7] proposed a framework for the DP with novel pruning techniques. Especially, it is experimentally shown that the DP with pruning is faster than both of the labeling algorithm and the ε-constraint method on 2-objective cases. Therefore, the DP with pruning is a better approach on the MKP.

In this chapter, we present more effective pruning techniques to the DP approach above by the TD-DD. An important idea is to use ZDDs for itemsets with knapsack-constraints (mentioned in Section 3.3.1) as a support data structure, which stores all the feasible solutions. The algorithm prunes more undesirable solutions than the existing approach. Moreover, the algorithm constructs a ZDD for Pareto-optimal solutions by an extended framework of the TD-DD with the new pruning techniques. Experimental results showed that the proposed algorithm is faster than the existing DP on various types of instances.

6.1 0-1 Multi-Objective Knapsack Problems

Let S be a set of n items $S := \{s_1, \ldots, s_n\}$ each of that $s_i \in S$ has an m-dimensional value vector $\boldsymbol{v}(s_i) \in \mathbb{Z}^m_{\geq 0}$ where $\boldsymbol{v}(s_i) = (v(s_i)_1, \ldots, v(s_i)_m) \neq \mathbf{0}$. For any subset $X \subseteq S$, let $\boldsymbol{v}(X) := \sum_{s_i \in X} \boldsymbol{v}(s_i)$. Here, we define the order relation $<$ between two m-dimensional vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}^m_{\geq 0}$ as

$$\boldsymbol{a} < \boldsymbol{b} \iff \forall i \in [m], a_i \leq b_i \text{ and } \exists j \in [m], a_j < b_j. \tag{6.1}$$

Similarly, its reversal order relation $>$ is also defined. We also define the partial order relation Δ, called the *dominance relation*, between two subsets $X, Y \subseteq S$ as

$$X \Delta Y \iff \boldsymbol{v}(X) < \boldsymbol{v}(Y). \tag{6.2}$$

If $X \Delta Y$, we say that X is dominated by Y, and Y dominates X.

Let w be a weight function $w\colon S \to \mathbb{N}$ and $w(X) := \sum_{s_i \in X} w(s_i)$ for any $X \subseteq S$. For a capacity $W \in \mathbb{N}$, let $\mathcal{X}_W := \{X \subseteq S \mid w(X) \leq W\}$, which is equal to the solution of the SEP P_{KP}. Given a capacity W, a subset $X \in \mathcal{X}_W$ is said to be *Pareto-optimal* if X is not dominated by any other subset $Y \in \mathcal{X}_W$.

A 0-1 multi-objective knapsack problem (MKP) requires obtaining all the subsets, called *Pareto-optimal solutions*, that are Pareto-optimal. Namely, an MKP is an SEP $P_{\mathrm{MK}} = \langle S, C_{\mathrm{MK}} \rangle$ where C_{MK} is defined as

$$C_{\mathrm{MK}}(X) = 1 \iff X \in \mathcal{X}_W, \neg(\exists Y \in \mathcal{X}_W, X \Delta Y). \tag{6.3}$$

6.2 Basic Framework

In this section, we introduce the existing dynamic programming framework proposed by Bazgan et al. [7] to solve MKPs. Subsequently, we propose a new framework based on the TD-DD that constructs a ZDD for $\mathcal{X}_{P_{\mathrm{MK}}}$. Let $S^{\geq i} := \{s_i, \ldots, s_n\}$.

6.2.1 Existing Framework

An existing framework is based on a dynamic programming. The algorithm maintains subsets by states each of that is a pair of a current total weight and a current value vector: Let λ be a state which has a weight $w(\lambda)$ and value vector $\boldsymbol{v}(\lambda)$. Two states λ, λ' are equivalent if $w(\lambda) = w(\lambda')$ and $\boldsymbol{v}(\lambda) = \boldsymbol{v}(\lambda')$.

The algorithm consists of n steps. Let $\mathcal{X}_i$ be a set of states at step $i \in [n]$. For convenience, let $\mathcal{X}_{n+1}$ be a set of states at the end of the algorithm. Initially, $\mathcal{X}_1 = \{\bar{\lambda}\}$ where $w(\bar{\lambda}) = 0$ and $\boldsymbol{v}(\bar{\lambda}) = \mathbf{0}$. At step $i \in [n]$, the algorithm conducts the following procedures: For each $\lambda \in \mathcal{X}_i$, the algorithm stores it with $\mathcal{X}_{i+1}$, and generate a new state λ^+ where $w(\lambda^+) = w(\lambda) + w(s_i)$ and $\boldsymbol{v}(\lambda^+) = \boldsymbol{v}(\lambda) + \boldsymbol{v}(s_i)$. If $w(\lambda^+) \leq W$, the algorithm store it with $\mathcal{X}_{i+1}$. Note that equivalent states are not stored with duplication.

At last of each step, the algorithm conducts a pruning procedure. For any state $\lambda \in \mathcal{X}_i$, let $\mathcal{E}_i(\lambda) \subseteq 2^{S^{\geq i}}$ be a set of extendable subsets where each $X \in \mathcal{E}_i(\lambda)$ satisfies $w(\lambda) + w(X) \leq W$. Then, the algorithm uses a relation Δ_i between states in $\mathcal{X}_{i+1}$ which satisfies

$$\lambda \Delta_i \lambda' \Longrightarrow \forall X \in \mathcal{E}_i(\lambda), \exists Y \in \mathcal{E}_i(\lambda'), \boldsymbol{v}(\lambda) + \boldsymbol{v}(X) < \boldsymbol{v}(\lambda') + \boldsymbol{v}(Y). \tag{6.4}$$

Let R_i be a set of such relations. For any state $\lambda \in \mathcal{X}_{i+1}$, if $\lambda \Delta_i \lambda'$ for a relation $\Delta_i \in R_i$ and a state $\lambda' \in \mathcal{X}_{i+1}$, the algorithm prune λ. Especially, the algorithm requires that at least one relation $\Delta_n^* \in R_n$ between states in $\mathcal{X}_{n+1}$ satisfies

$$\boldsymbol{v}(\lambda) < \boldsymbol{v}(\lambda') \Longrightarrow \lambda \Delta_n^* \lambda'. \tag{6.5}$$

Then, the set of remaining states in $\mathcal{X}_{n+1}$ corresponds to all the Pareto-optimal solutions. The pseudo code of the framework is shown in Algorithm 6.2.1. Bazgan et al. proposed some relations for pruning in the literature [7].

6.2.2 Proposed Framework

We propose a framework that is a transformation of the TD-DD in accordance to the existing framework above. A main difference between proposed framework and the TD-DD is that the framework does not use $\bot$-prune function and $\top$-prune function. The algorithm conducts a pruning procedure at last of each step instead.

Algorithm 6.2.1 Existing framework for solving MKPs

$\mathcal{X}_1 \leftarrow \{\bar{\lambda}\}$ where $w(\bar{\lambda}) = 0$ and $\boldsymbol{v}(\bar{\lambda}) = \mathbf{0}$
$\mathcal{X}_i \leftarrow \emptyset$ for $i = 2, \dots, n+1$
for $i = 1, \dots, n$ **do**
  for $\lambda \in \mathcal{X}_i$ **do**
    $\mathcal{X}_{i+1} \leftarrow \mathcal{X}_{i+1} \cup \{\lambda\}$
    Generate λ^+
    if $w(\lambda^+) \leq W$ **then**
      $\mathcal{X}_{i+1} \leftarrow \mathcal{X}_{i+1} \cup \{\lambda^+\}$
    end if
  end for
  for $\lambda \in \mathcal{X}_{i+1}$ **do**
    if $\exists \Delta_i \in R_i, \exists \lambda' \in \mathcal{X}_{i+1}, \lambda \Delta_i \lambda'$ **then**
      $\mathcal{X}_{i+1} \leftarrow \mathcal{X}_{i+1} \setminus \{\lambda\}$
    end if
  end for
end for
return $\mathcal{X}_{n+1}$

As a preprocess, we obtain a ZDD $\hat{Z}$ for $\mathcal{X}_W$ by the TD-DD in Section 3.3.1 and reduce it. After that, the algorithm processes the items $s_1, \dots, s_n$ in this order, updates the configuration, and merges equivalent nodes as with the TD-DD. Let $\hat{N}$ be the node set of $\hat{Z}$ and $\hat{\rho}$ be the root node of $\hat{Z}$. Let $\mathcal{D} = (N, A)$ be an initial decision diagram such that $N = \{\rho, \bot, \top\}$ and $A = \emptyset$.

We define the configuration as the pair of a node in $\hat{N}$ and a current value vector:

$$\phi(\alpha) := (\tau(\alpha), \boldsymbol{v}(\alpha)) \tag{6.6}$$

where $\tau(\alpha) \in \hat{N}$, $\ell(\alpha) = \ell(\tau(\alpha))$, and $\boldsymbol{v}(\alpha) = \boldsymbol{v}(X)$ for all $X \in \bar{\mathcal{D}}(\alpha)$. For all generated node α, the algorithm stores it with the node set $N_{\ell(\tau(\alpha))}$. Initially, $\tau(\rho) = \hat{\rho}$ and $\boldsymbol{v}(\rho) = \mathbf{0}$. For any $\alpha \in N_i$ and $b \in \{0, 1\}$, generateNode(α, s_i, b) computes the configuration of α_b as follows: $\tau(\alpha_b) = \tau(\alpha)_b$ and $\boldsymbol{v}(\alpha_b) = \boldsymbol{v}(\alpha) + b \times \boldsymbol{v}(s_i)$. Note that the current total weight is not explicitly maintained because it is implicitly maintained by $\tau(\alpha)$: For all $X \in \bar{\mathcal{D}}(\alpha)$, $X \notin \mathcal{X}_W$ iff $\tau(\alpha) = \bot$ because X is represented by a path $\hat{\rho}$ to $\bot$ on $\hat{\mathcal{Z}}$.

At last of each step $i \in [n]$, the algorithm conducts a pruning procedure as follows: First, any $\alpha \in N_{i+1}$ is changed into $\bot$ if $\tau(\alpha) = \bot$. Subsequently, we consider relations, which are an extension of (6.5), between remaining nodes. Let R be a set of relations, called *future dominance relations* (FDRs), between nodes such that each $\hat{\Delta} \in R$ satisfies

$$\alpha \hat{\Delta} \beta \Longrightarrow \forall X \in \hat{\mathcal{Z}}(\tau(\alpha)), \exists Y \in \hat{\mathcal{Z}}(\tau(\beta)), \boldsymbol{v}(\alpha) + \boldsymbol{v}(X) < \boldsymbol{v}(\beta) + \boldsymbol{v}(Y). \tag{6.7}$$

A set family $\hat{\mathcal{Z}}(\alpha)$ corresponds to a set of extendable subsets in the existing framework. If $\alpha\hat{\Delta}\beta$, we say that α is dominated by β, and β dominates α. For each FDR $\hat{\Delta} \in R$, the algorithm changes α into $\perp$ for any $\alpha \in N_{i+1}$ which is dominated by another node. Especially, the algorithm requires that at least one FDR $\Delta^* \in R$ satisfies

$$\ell(\alpha) = n+1, \exists Y \in \hat{\mathcal{Z}}(\tau(\beta)), \boldsymbol{v}(\alpha) < \boldsymbol{v}(\beta) + \boldsymbol{v}(Y). \Longrightarrow \alpha\Delta^*\beta \tag{6.8}$$

Then, all the Pareto-optimal solutions are represented by a path from ρ to a remaining node in N_{n+1}. Thus, all the remaining nodes in N_{n+1} are changed into $\top$. The pseudo code of the framework is shown in Algorithm 6.2.2.

6.3 Future Dominance Relations Using ZDDs

In this section, we design two FDRs for pruning. They are based on the capacity and the tight upper bound, respectively, each of which can be computed by using $\hat{\mathcal{Z}}$.

Relation Based on Implicit Capacity

For any node $\gamma \in \hat{N}$, we define the implicit capacity $IC(\gamma)$ as

$$IC(\gamma) := \max_{X \in \hat{\mathcal{Z}}(\gamma)} w(X). \tag{6.9}$$

Here, we have the following lemma.

Lemma 6.3.1. *Any pair of nodes $\alpha, \beta \in N$ satisfies*

$$\begin{aligned} &IC(\tau(\alpha)) \leq IC(\tau(\beta)) \text{ and } \boldsymbol{v}(\alpha) < \boldsymbol{v}(\beta) \\ &\Longrightarrow \forall X \in \hat{\mathcal{Z}}(\tau(\alpha)), \exists Y \in \hat{\mathcal{Z}}(\tau(\beta)), \boldsymbol{v}(\alpha) + \boldsymbol{v}(X) < \boldsymbol{v}(\beta) + \boldsymbol{v}(Y), \end{aligned} \tag{6.10}$$

and

$$\ell(\alpha) = n+1 \Longrightarrow IC(\tau(\alpha)) \leq IC(\tau(\beta)). \tag{6.11}$$

Proof. Obviously, $IC(\tau(\alpha)) \leq IC(\tau(\beta))$ implies $\hat{\mathcal{Z}}(\tau(\alpha)) \subseteq \hat{\mathcal{Z}}(\tau(\beta))$. Thus, for any subset $X \in \hat{\mathcal{Z}}(\tau(\alpha))$, there is a subset $Y \in \hat{\mathcal{Z}}(\tau(\beta))$ where $\boldsymbol{v}(X) = \boldsymbol{v}(Y)$ or $\boldsymbol{v}(X) < \boldsymbol{v}(Y)$. Especially, if $\ell(\alpha) = n+1$, then $IC(\tau(\alpha)) = 0 \leq IC(\tau(\beta))$. □

By the above lemma, we define a FDR Δ_{IC}, which satisfies the conditions (6.7) and (6.8), as

$$\alpha\Delta_{IC}\beta \iff IC(\tau(\alpha)) \leq IC(\tau(\beta)) \text{ and } \boldsymbol{v}(\alpha) < \boldsymbol{v}(\beta). \tag{6.12}$$

Relation Based on Tight Upper Bound

For any node $\gamma \in \hat{N}$, we define the greedy extension $\boldsymbol{g}(\gamma)$ by the following procedures: Initially, $X = \emptyset$. We start the descent from γ and finish at $\top$. Let β is the current node. We update $X \leftarrow X \cup \{s_{\ell(\beta)}\}$, and descend 1-arc of β. Finally, $\boldsymbol{g}(\gamma) := \boldsymbol{v}(X)$. Note that, because $\hat{\mathcal{Z}}$ is a reduced ZDD, any 1-arc does not point $\bot$.

For any node $\gamma \in \hat{N}$, let $\boldsymbol{u}(\gamma)$ be an m-dimensional vector , which is the tight upper bound of $\hat{\mathcal{Z}}(\gamma)$, defined as

$$\boldsymbol{u}(\gamma) := \left(\max_{X \in \hat{\mathcal{Z}}(\gamma)} v(X)_1, \ldots, \max_{X \in \hat{\mathcal{Z}}(\gamma)} v(X)_m \right). \tag{6.13}$$

Here, we have the following lemma.

Lemma 6.3.2. *Any pair of nodes $\alpha, \beta \in N$ satisfies*

$$\begin{aligned} &\boldsymbol{v}(\alpha) + \boldsymbol{u}(\tau(\alpha)) < \boldsymbol{v}(\beta) + \boldsymbol{g}(\tau(\beta)) \\ &\Longrightarrow \forall X \in \hat{\mathcal{Z}}(\tau(\alpha)), \exists Y \in \hat{\mathcal{Z}}(\tau(\beta)), \boldsymbol{v}(\alpha) + \boldsymbol{v}(X) < \boldsymbol{v}(\beta) + \boldsymbol{v}(Y), \end{aligned} \tag{6.14}$$

and

$$\ell(\alpha) = n+1 \Longrightarrow \boldsymbol{u}(\tau(\alpha)) = \boldsymbol{g}(\tau(\beta)) \text{ or } \boldsymbol{u}(\tau(\alpha)) < \boldsymbol{g}(\tau(\beta)). \tag{6.15}$$

Proof. By the definition, $\boldsymbol{v}(X) = \boldsymbol{u}(\alpha)$ or $\boldsymbol{v}(X) < \boldsymbol{u}(\alpha)$ for any subset $X \in \hat{\mathcal{Z}}(\tau(\alpha))$. In addition, there is a subset $Y \in \hat{\mathcal{Z}}(\tau(\beta))$ where $\boldsymbol{v}(Y) = \boldsymbol{g}(\tau(\beta))$. Especially, if $\ell(\alpha) = n+1$, then $\boldsymbol{u}(\alpha) = \boldsymbol{0}$. Thus, the lemma follows. □

By the above lemma, we define a FDR Δ_u, which satisfies the conditions (6.7) and (6.8), as

$$\alpha \Delta_u \beta \iff \boldsymbol{v}(\alpha) + \boldsymbol{u}(\tau(\alpha)) < \boldsymbol{v}(\beta) + \boldsymbol{g}(\tau(\beta)). \tag{6.16}$$

6.4 Algorithm with Efficient Techniques

By the above section, we can apply the proposed framework to MKPs. However, naive procedures make the computation time longer. Therefore, in this section, we present some techniques to reduce the computation time practically.

6.4.1 Preprocess

To evaluate the future dominance relations Δ_{IC} and Δ_u efficiently, we precompute $IC(\alpha)$, $\boldsymbol{g}(\alpha)$, and $\boldsymbol{u}(\alpha)$ for all nodes $\alpha \in \hat{N}$. Because the computation of $IC(\alpha)$ and $\boldsymbol{u}$ is

a maximization with respect to w and v_i ($i \in [m]$), we precompute them by the bottom-up dynamic programming like the Algorithm 2.4.10. Similarly, $\boldsymbol{g}(\alpha)$ can be precomputed by a bottom-up procedure. The pseudo code of the preprocess is shown in the Algorithm 6.4.1. Note that we use the notation $\hat{N}_i := \{\alpha \in \hat{N} \mid \ell(\alpha) = i\}$.

6.4.2 Pruning

The pruning procedures should be accelerated as much as possible because it requires the highest computation time: A naive procedure is to compare all the pairs of nodes in N_{i+1}, which requires $O(m|N_{i+1}|^2)$ amount of time. Here, we present some techniques for accelerating the naive pruning procedures. A main effect of the techniques is to reduce the candidates for the comparison.

Pruning with Δ_{IC}

We define the reversal lexicographical order $\geq_{lex}$ on m-dimensional vectors as

$$\boldsymbol{a} \geq_{lex} \boldsymbol{b} \iff \exists i \in [m], a_i > b_i, \forall j \in [i-1], a_j = b_j \text{ or } \boldsymbol{a} = \boldsymbol{b}. \tag{6.17}$$

Note that $\boldsymbol{a} > \boldsymbol{b}$ implies $\boldsymbol{a} \geq_{lex} \boldsymbol{b}$. We also define an order relation $\geq_{IC}$ on N_i as

$$\alpha \geq_{IC} \beta \iff IC(\tau(\alpha)) > IC(\tau(\beta)) \text{ or } IC(\tau(\alpha)) = IC(\tau(\beta)), \boldsymbol{v}(\alpha) \geq_{lex} \boldsymbol{v}(\beta). \tag{6.18}$$

Here, we have the following lemma.

Lemma 6.4.1. $\alpha \Delta_{IC} \beta \Longrightarrow \beta \geq_{IC} \alpha$.

Proof. Immediately consequence of the definitions. □

Because $\geq_{IC}$ is a total order, we can sort all the nodes in N_i in the decreasing order of $\geq_{IC}$. We can efficiently conduct the pruning with Δ_{IC} for each node $\alpha \in N_i$ in the sorted order as follows: The algorithm maintains a list of m-dimensional vectors DL that stores candidates for the comparison in the decreasing order of $\geq_{lex}$. Initially, DL is empty. Subsequently, for each node $\alpha \in N_i$, the algorithm conducts the following procedures. Let the current DL store the k vectors $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k$ in this order. For convenience, let $\boldsymbol{d}_{k+1} := \boldsymbol{0}$.

1. Let j be the current index on DL. Initially, $j = 1$.
2. While $\boldsymbol{d}_j \geq_{lex} \boldsymbol{v}(\alpha)$, we have a chance to prune α by the definition of $\geq_{lex}$: If $\boldsymbol{v}(\alpha) < \boldsymbol{d}_j$, prune α and finish the procedure. If $\boldsymbol{v}(\alpha) = \boldsymbol{d}_j$, finish the procedure. Otherwise, j is updated by $j+1$.

3. Insert $\boldsymbol{v}(\alpha)$ into the j-th position of DL.

4. Delete all the vectors $\boldsymbol{d}_l$ where $\boldsymbol{d}_l < \boldsymbol{v}(\alpha)$ for $l = j, \ldots, k$.

After the procedures, all the vectors in DL are sorted in the decreasing order of $\geq_{lex}$. Let δ be the maximum cardinality of DL until the end of the pruning with Δ_{IC}. The time complexity of the pruning with Δ_{IC} is $O(m\delta|N_i|)$ where $\delta \leq |N_i|$.

Pruning with Δ_u

Similarly, we can efficiently conduct the pruning with Δ_u as follows: The algorithm constructs a list of m-dimensional vectors GL that stores candidates for the comparison in the decreasing order of $\geq_{lex}$. Initially, GL is empty. Subsequently, for each node $\alpha \in N_i$, the algorithm conducts the following procedures which are similar to the pruning with Δ_{IC}. Let the current GL store the k vectors $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_k$ in this order. For convenience, let $\boldsymbol{g}_{k+1} := \boldsymbol{0}$. We use the notation $\boldsymbol{t}(\alpha) := \boldsymbol{v}(\alpha) + \boldsymbol{g}(\tau(\alpha))$ for short.

1. Let j be the current index on GL. Initially, $j = 1$.

2. While $\boldsymbol{g}_j \geq_{lex} \boldsymbol{t}(\alpha)$, we check whether $\boldsymbol{t}(\alpha)$ is unnecessary or not for the comparison: If $\boldsymbol{t}(\alpha) < \boldsymbol{g}_j$ or $\boldsymbol{t}(\alpha) = \boldsymbol{g}_j$, finish the procedure, namely, $\boldsymbol{t}(\alpha)$ is unnecessary for the comparison. Otherwise, j is updated by $j + 1$.

3. Insert $\boldsymbol{t}(\alpha)$ into the j-th position of GL.

4. Delete all the vectors $\boldsymbol{g}_l$ where $\boldsymbol{g}_l < \boldsymbol{t}(\alpha)$ for $l = j, \ldots, k$.

After the procedures, all the vectors in GL are sorted in the decreasing order of $\geq_{lex}$. Next, for each node $\alpha \in N_i$, the algorithm conducts the pruning with Δ_u by using GL as follows:

1. Let j be the current index on GL. Initially, $j = 1$.

2. While $\boldsymbol{g}_j \geq_{lex} \boldsymbol{v}(\alpha) + \boldsymbol{u}(\tau(\alpha))$, we have a chance to prune α: If $\boldsymbol{v}(\alpha) + \boldsymbol{u}(\tau(\alpha)) < \boldsymbol{g}_j$, prune α and finish the procedure. Otherwise, j is updated by $j + 1$.

Let $\hat{\delta}$ be the maximum cardinality of GL until the end of the pruning with Δ_u. The time complexity of the pruning with Δ_u is $O(m\hat{\delta}|N_i|)$ where $\hat{\delta} \leq |N_i|$.

6.4.3 Item Reordering Heuristics

The order of items is an important issue in the single-objective knapsack problem. As a well known result, decreasing order with respect to $\frac{v(s_i)_1}{w(s_i)}$, i.e., value per unit weight, is better. For the MKP, Bazgan et al. proposed the order $\mathcal{O}^{max}$, which refers to the ranking of items with respect to each objective $v(s_i)_j$. For details, please refer to [7].

Here, we propose a new order which is a natural expansion of single-objective cases as follows: Let $\boldsymbol{p}(s_i)$ be the *potential vector* of an item $s_i \in S$ where

$$\boldsymbol{p}(s_i) := \left(\frac{v(s_i)_1}{w(s_i)}, \ldots, \frac{v(s_i)_m}{w(s_i)} \right). \tag{6.19}$$

We define $\mathcal{O}^{pot}$ as the decreasing order of the potential vectors on $\geq_{lex}$. Moreover, we define $\mathcal{O}^{\widetilde{pot}}$ as the reverse order of $\mathcal{O}^{pot}$.

6.5 Experiments

All the code was implemented in C++ (g++5.4.0 with the -O3 option). We used 64-bit Ubuntu 16.04 LTS with an Intel Core i7-3930K 3.2 GHz CPU and 64 GB RAM. All the instances satisfy the condition that $10 \leq v(s_i)_j, w(s_i) \leq 100$ for each $s_i \in S$ and each $j \in [m]$. The capacities of all the instances are $W = 10n$. The following three types of instances were considered.

- Type 1: All the values and weights are decided uniformly at random. Type 1 is an easier setting than the following two types.
- Type 2: For each $s_i \in S$, its value vector satisfies the condition that $50m - 10 \leq \sum_{j=1}^{m} v(s_i)_j \leq 50m + 10$. This causes a negative correlation between the objectives which renders it difficult to solve instances. All the weights are decided uniformly at random.
- Type 3: All values are decided uniformly at random. For each $s_i \in S$, its weight satisfies the condition that $\frac{1}{m}\sum_{j=1}^{m} v(s_i)_j - 10 \leq w(s_i) \leq \frac{1}{m}\sum_{j=1}^{m} v(s_i)_j + 10$. This causes a positive correlation between the values and weights. Type 3 is the most difficult setting among the aforementioned three types.

6.5.1 Evaluation of Item Reordering Heuristics

First, to observe the effect of reordering heuristics, we compared the three reordering heuristics ($\mathcal{O}^{max}$, $\mathcal{O}^{pot}$, and $\mathcal{O}^{\widetilde{pot}}$) and random order on 100 instances of each type. Table 6.1 indicates that $\mathcal{O}^{pot}$ accelerates the proposed algorithm. However, the computation

Table 6.1. Average computation time (sec) of each reordering

			Bazgan et al.'s Method				Proposed Method			
Type	n	m	random	$\mathcal{O}^{max}$	$\mathcal{O}^{pot}$	$\widetilde{\mathcal{O}^{pot}}$	random	$\mathcal{O}^{max}$	$\mathcal{O}^{pot}$	$\widetilde{\mathcal{O}^{pot}}$
	200	2	5.31	**3.05**	4.32	9.79	10.78	5.81	**4.35**	12.96
1	100	3	**18.42**	19.64	19.17	59.57	15.98	11.85	**7.59**	28.92
	70	4	**19.20**	29.03	21.89	81.88	9.84	7.34	**5.59**	30.77
	150	2	7.16	6.15	**6.02**	8.42	11.86	10.37	**6.63**	9.99
2	60	3	**10.81**	16.06	12.94	20.47	5.84	7.78	**3.69**	10.12
	50	4	**70.17**	105.25	**70.17**	126.93	17.12	25.08	**9.71**	40.93
	120	2	18.37	**9.24**	11.60	9.89	35.31	19.55	18.82	**18.22**
3	50	3	52.91	40.81	41.47	**36.29**	31.30	23.38	**20.95**	22.83
	35	4	33.39	29.91	29.26	**26.27**	7.50	6.43	**5.96**	6.71

time of the previous algorithm proposed by Bazgan et al. did not vary considerably with reordering heuristics. Thus, in the following experiments, we used $\mathcal{O}^{pot}$ for reordering items.

6.5.2 Performance on Uniformly Random Instances

Subsequently, to compare the basic performance of the previous algorithm proposed by Bazgan et al. with that of the proposed algorithm, we investigated the average computation time and the number of states generated by each algorithm for instances of Type 1. The two-objective ($m = 2$) cases have $n = 25 \times k$ items. The three-objective ($m = 3$) cases have $n = 10 \times k$ items. The four-objective ($m = 4$) cases have $n = 10 \times k$ items. For each pair of parameters n and m, we used 100 instances. The results are shown in Fig.6.1.

For the two-objective cases, the number of states generated by the proposed algorithm was approximately half that generated by the previous algorithm. However, the difference between the computation times of both algorithms was slight. The reason is that the effect of the overhead of constructing ZDDs in the proposed algorithm appears greater than that of the pruning process, because two-objective cases are relatively easy to solve.

For the three-objective cases, the proposed algorithm was up to two times faster and generated up to four times fewer states than the previous algorithm.

Moreover, the differences in the computation times and numbers of states were considerably larger for the four-objective cases. The proposed algorithm was up to four times faster and generated up to six times fewer states than the previous algorithm. This

shows that the proposed algorithm is more effective than the previous algorithm as the number of objectives increases.

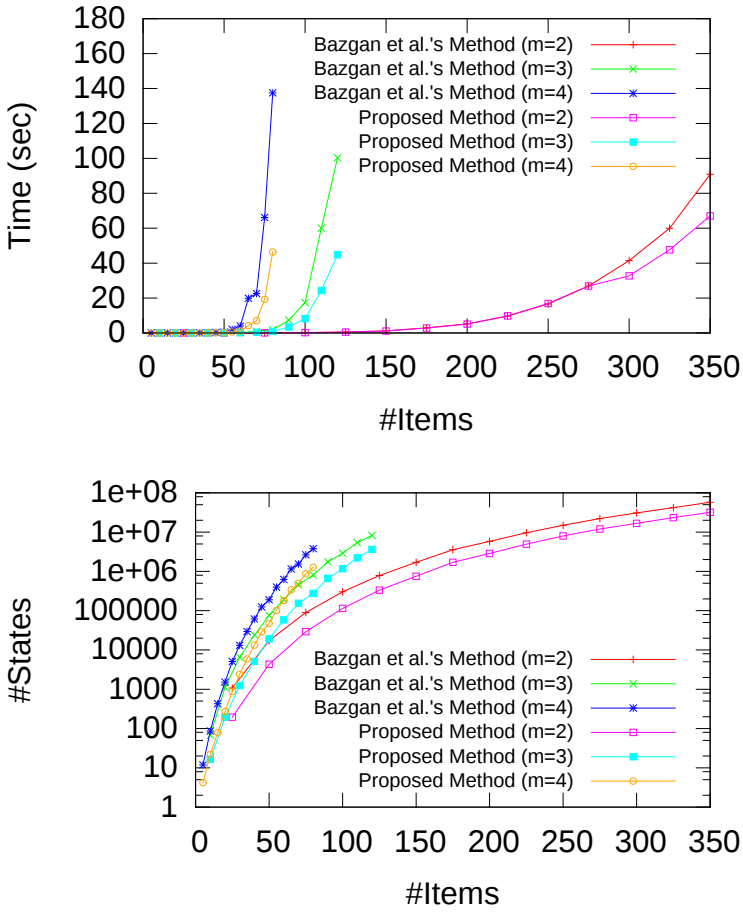


Figure 6.1. Computational results for instances of Type 1.

6.5.3 Performance on Correlated Random Instances

Next, to evaluate the performance of the proposed algorithm in detail, we conducted an experiment with correlated random instances of Type 2 and 3. These instances are relatively difficult to solve. We investigated the average computation time and number of states generated by each algorithm.

Type 2

The settings of instances of Type 2 are as follows. The two-objective ($m = 2$) cases have $n = 50 \times k$ items. The three-objective ($m = 3$) cases have $n = 15 \times k$ items. The

four-objective ($m = 4$) cases have $n = 10 \times k$ items. For each pair of parameters n and m, we used 100 instances.

The results for the instances of Type 2 are shown in Fig.6.2. For the two-objective cases, the proposed algorithm generated fewer states than the previous algorithm but the two algorithms required approximately equal computation times. These results are similar to those obtained for random instances.

However, the differences between the two algorithms were revealed in the three- and four-objective cases. The proposed algorithm always performed better than the previous algorithm. It was approximately three and six times faster, respectively, and generated approximately three and four times fewer states, respectively, for three- and four-objective cases in comparison with the previous algorithm. This suggests that the number of objectives is crucial for the efficiency of the proposed algorithm in comparison with the previous algorithm.

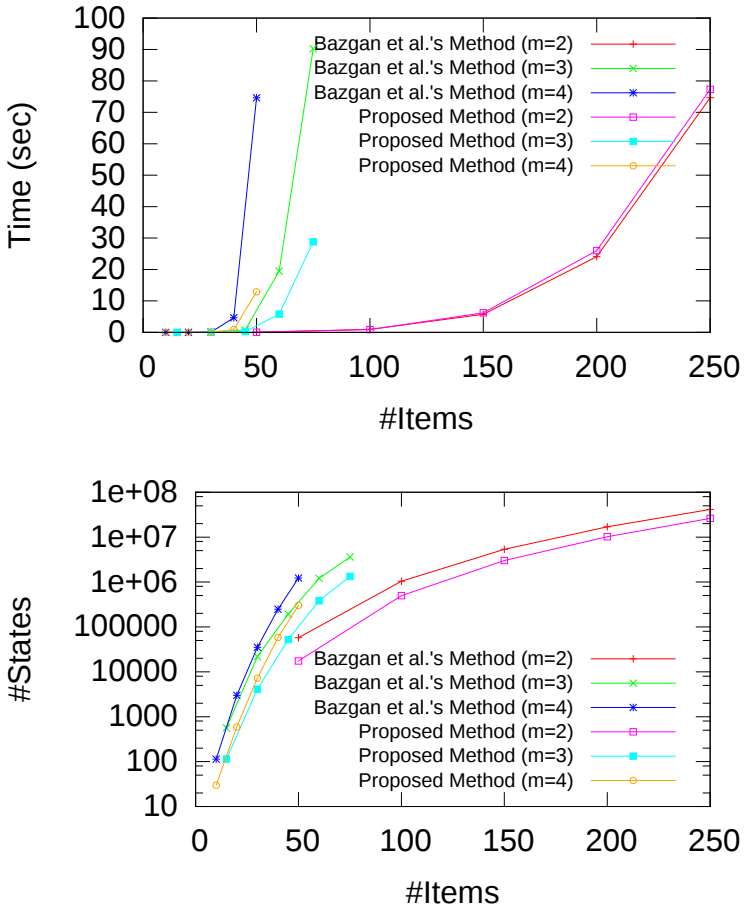


Figure 6.2. Computational results for instances of Type 2.

Table 6.2. Average number of Pareto-optimal solutions and nodes of output ZDDs

	$m=2$			$m=3$			$m=4$		
Type	n	#Pareto	#Nodes	n	#Pareto	#Nodes	n	#Pareto	#Nodes
1	350	450	7793	120	1645	5451	80	2976	5126
2	250	651	8142	75	2758	4873	50	4196	3670
3	150	748	5767	50	3108	4482	40	10482	8765

Type 3

The settings of instances of Type 3 are as follows. The two-objective ($m = 2$) cases have $n = 25 \times k$ items. The three-objective ($m = 3$) cases have $n = 10 \times k$ items. The four-objective ($m = 4$) cases have $n = 10 \times k$ items. For each pair of parameters n and m, we used 100 instances.

For the instances of Type 3, the results are shown in Fig.6.3. For the two-objective cases, the proposed algorithm generated fewer states than the previous algorithm; however, it was up to two times slower. Its computation time was up to two times longer. These results show that the overhead of constructing ZDDs in the proposed algorithm is predominant in this type of instances.

However, the proposed algorithm performed better than the previous algorithm in three- and four-objective cases. It was approximately two and four times faster, respectively, and generated approximately three and four times fewer states, respectively, for three- and four-objective cases in comparison with the previous algorithm. These results show that the proposed algorithm is efficient for solving difficult instances.

6.5.4 Evaluation of Constructed ZDDs

Finally, we investigated the number of Pareto-optimal solutions and the size of the output ZDDs for each type. The results are shown in Table 6.2. They suggest that the number of Pareto-optimal solutions suddenly increases as the objectives increase and the instances become difficult to solve.

Each Pareto-optimal solution has approximately $\frac{n}{5}$ items depending on the method of generation of instances. Thus, the size of output ZDDs was slightly large in the two-objective cases that had fewer Pareto-optimal solutions than the other cases. However, for the three- and four-objective cases, the ZDDs were sufficiently small for the total number of items in all the Pareto-optimal solutions.

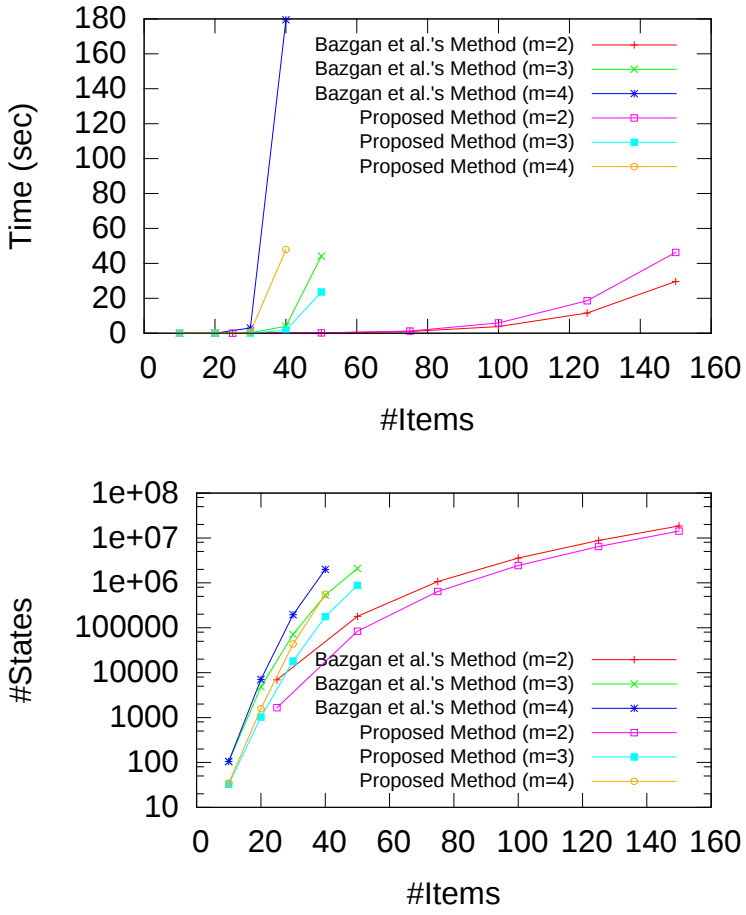


Figure 6.3. Computational results for instances of Type 3.

6.6 Concluding Remarks for 0-1 Multi-Objective Knapsack Problems

In this chapter, we proposed an algorithm to implicitly enumerate Pareto-optimal solutions of the MKP using the TD-DD of the ZDD. The algorithm uses the TD-DD in the preprocess. Subsequently, the algorithm uses an extended framework of the TD-DD derived from an existing DP algorithm with novel pruning techniques. An important idea of the algorithm is the new pruning techniques that utilizes properties of ZDDs. The experimental results showed that the proposed algorithm was faster than the existing algorithm on various types of three- and four-objective instances. We also confirmed that the sizes of the constructed ZDDs are rather small.

An important direction of future work is to consider whether main ideas in this chapter can be applied to MOCOs with linear criteria or not. It is also important to apply ZDDs representing Pareto-optimal solutions to real-world issues.

Algorithm 6.2.2 Constructing a ZDD for $\mathcal{X}_{P_{\mathrm{MK}}}$

Precompute $\hat{\mathcal{Z}}$ for $\mathcal{X}_W$
$N_1 \leftarrow \{\rho\}$ where $\tau(\rho) = \hat{\rho}$ and $\boldsymbol{v}(\rho) = \mathbf{0}$
$N_i \leftarrow \emptyset$ for $i = 2, \ldots, n+1$
Generate the terminals $\bot$ and $\top$.
$A_b \leftarrow \emptyset$ for each $b \in \{0,1\}$
for $i = 1, \ldots, n$ **do**
  for $\alpha \in N_i$ **do**
    for $b \in \{0,1\}$ **do**
      $\beta \leftarrow \mathrm{generateNode}(\alpha, s_i, b)$
      if $\tau(\beta) = \bot$ **then**
        $\beta \leftarrow \bot$
      else
        if $\exists \beta' \in N_{\ell(\tau(\beta))},\ \phi(\beta) = \phi(\beta')$ **then**
          $\beta \leftarrow \beta'$
        else
          $N_{\ell(\tau(\beta))} \leftarrow N_{\ell(\tau(\beta))} \cup \{\beta\}$
        end if
      end if
      $A_b \leftarrow A_b \cup \{(\alpha, \beta)\}$
    end for
  end for
  for $\alpha \in N_{i+1}$ **do**
    if $\exists \Delta^* \in R, \exists \beta \in N_{i+1}, \alpha \Delta^* \beta$ **then**
      Change α into $\bot$
    end if
  end for
end for
for $\alpha \in N_{n+1}$ **do**
  Change α into $\top$
end for
$N \leftarrow (\bigcup_{i=1,\ldots,m} N_i) \cup \{\bot, \top\}$, $A \leftarrow A_0 \cup A_1$
return $\mathcal{D} = (N, A)$

Algorithm 6.4.1 Preprocess

1: $IC(\bot) \leftarrow 0$, $IC(\top) \leftarrow 0$, $\boldsymbol{g}(\top) \leftarrow \mathbf{0}$, $\boldsymbol{u}(\bot) \leftarrow \mathbf{0}$, $\boldsymbol{u}(\top) \leftarrow \mathbf{0}$
2: **for** $i = n, \ldots, 1$ **do**
3: **for** $\alpha \in \hat{N}_i$ **do**
4: $IC(\alpha) \leftarrow \max\{IC(\alpha_0), IC(\alpha_1) + w(s_i)\}$
5: $\boldsymbol{g}(\alpha) \leftarrow \boldsymbol{g}(\alpha_1) + \boldsymbol{v}(s_i)$
6: $u(\alpha)_j \leftarrow \max\{u(\alpha_0)_j, u(\alpha_1)_j + v(s_i)_j\}$ for $j = 1, \ldots, m$
7: **end for**
8: **end for**

Chapter 7

Conclusions and Open Problems

In this thesis, we dealt with the implicit enumeration to solve combinatorial problems efficiently avoiding the combinatorial explosion. As an approach to the implicit enumeration, we used the BDD and the ZDD which are a compact representation of set families. The BDD/ZDD has various useful functions such as set operations (family algebra), counting the number of subsets, computation of occurrence probability, and optimization with linear criteria. In addition, we focused on the FBS which is a top-down construction framework of BDDs/ZDDs for constrained edge subsets on graphs.

In Chapter 3, we presented a generalized framework of the FBS, named the TD-DD, for a top-down construction of BDDs/ZDDs on various combinatorial structures. The TD-DD has four main components configuration, generateNode function, $\bot$-prune function, and $\top$-prune function. We should design the components adequately for the implicit enumeration of desirable subsets. As examples, we introduced the TD-DD for itemsets with knapsack-constraints and degree constrained subgraphs.

Subsequently, we showed usefulness of the TD-DD for various combinatorial problems: the problems are related to various combinatorial structures such as feasible mine assignments of Minesweeper puzzles, SCSSs on directed graphs, and Pareto-optimal solutions of MKPs. As an important contribution, we proposed novel algorithms based on the TD-DD for the implicit enumeration of the combinatorial structures above. We also conducted computational experiments to show the performances of our algorithms.

In Chapter 4, we applied the TD-DD for the Minesweeper puzzle: We defined a problem called the MGP which requires obtaining all the feasible mine assignments on a given fixed board. Subsequently, we proposed two approaches using ZDDs. One is a naive combinatorial approach combined with the family algebra. The other gives the MGP another formulation by degree constraints on graphs and uses the TD-DD for degree constrained subgraphs. We also showed that our approach can be applied to the MCP, the #M, the #MC, and the computation of mine probability. The experimental results showed that the approach using the TD-DD is better on various instances.

In Chapter 5, we dealt with strongly connected spanning subgraphs on directed graphs: We proposed an algorithm for the implicit enumeration of SCSSs on a given directed graphs. The algorithm is based on the TD-DD of BDDs. We also applied the algorithm to computing the SCR and obtaining the min-SCSS which are useful for various issues on network systems. The experimental results showed that the algorithm ran on various synthetic graphs and real-world graphs with a few hundred edges. Moreover, we succeeded the exact computation of the SCRs on real-world graphs above, which was previously impossible.

In Chapter 6, we dealt with Pareto-optimal solutions of the MKP: We proposed an algorithm to solve the MKP using the TD-DD. Characteristic things of the algorithm are to use the TD-DD for knapsack-constraints and linear optimization on the ZDD as support techniques. Especially, the algorithm uses extended framework of the TD-DD derived from the existing algorithm. An important contribution is to propose the new pruning techniques utilizing the ZDD. In addition, we presented a new heuristic for item reordering, which accelerates our algorithm. The experimental results showed that our algorithm was faster than the existing algorithm on various types of instances.

Thus, we succeeded to effectively apply the TD-DD for various combinatorial problems on puzzles, graphs, and itemsets. However, the proposed algorithms cannot run on large-scale instances. It is a significant issue of this research. In addition, we only dealt with combinatorial structures which are a part of *discrete structures*, i.e., permutations, strings, and the rest. Therefore, the author now explores the following future directions.

- Applying the TD-DD to other types of decision diagrams for the implicit enumeration of various discrete structures: Types of decision diagrams are not only the BDD/ZDD but also the *multi-valued decision diagrams* (MDD) [31], the *permutation decision diagrams* (π-DD) [48], the *rotation-based permutation decision diagrams* (Rot-π-DD) [27], the *sequence decision diagrams* (SeqDD) [42], and the rest. The MDD has some additional arcs for each node which can support to represent a set of tuples of subsets. The π-DD represents a set of permutations, and the Rot-π-DD is a derivation of the π-DD with different representation rules. The Seq-BDD represents a set of strings. The author would like to apply the TD-DD to other types of decision diagrams above for the implicit enumeration of various discrete structures.

- Implicit partial enumeration of large-scale significant SCSSs: In Chapter 5, we succeeded in dealing with small-scale graphs. On the other hand, various real-world issues tend to involve more large-scale instances recently due to the growth of the information technology. However, because large-scale subsets occur increase of the BDD size, dealing with large-scale instances is difficult for our current approach. Therefore, we need a novel approach reducing the BDD size. One

of such approach is to accept omission on the implicit enumeration, namely, we obtain a subset of desirable set family by constructing approximated BDDs. An important issue of the approach is to efficiently avoid omission of significant subsets which causes clear errors in applications such as computation of occurrence probability and optimization. For solving the issue, the *restrict* approach for some problems on undirected graphs has been studied in the literature [9], which prunes subsets guessed to have a little effect for the objective. The author considers applying the approach to our algorithm.

- Applications of the framework for solving the MKP in Chapter 6: An important characteristic of the framework is the procedure of pruning dominated nodes after generating all child nodes at each step. The author guesses that the procedure can be extended for dealing with constraints affecting between subsets, for example, considering priority of subsets like the Pareto-optimality. The author also guesses that the framework can be used for obtaining approximated Pareto-optimal solutions on large-scale instances of the MKP, because the existing algorithm in the literature [7] has been extended for such purpose [6].

As shown in this thesis, the implicit enumeration by the TD-DD can be a practical method for combinatorial problems, and we have future work for various directions. Thus, we believe that new useful techniques for the implicit enumeration based on the TD-DD are developed to solve various combinatorial problems and real-world issues.

Acknowledgements

First of all, I would like to thank Professor Shin-ichi Minato. He was my supervisor Professor until he was transferred to Kyoto University and I became a third year doctoral student. He always gave me advice for new interesting insights in my research topics. His rich knowledge helped me in not only my research but also daily life.

I also would like to thank Hiroki Arimura, the Professor at Information Knowledge Network Laboratory. He supervised me for a long while and became my supervisor Professor on behalf of Professor Minato after I became a third year doctoral student. He centrally advised me on important matters to take a doctoral degree. I also would like to thank Masaharu Yoshioka, the Professor at Knowledgebase Laboratory, and Ichigaku Takigawa, the Associate Professor at Large-Scale Knowledge Processing Laboratory, for their valuable comments and supports to write this thesis.

I also would like to express my gratitude to my co-authors Takashi Horiyama, Ryo Yoshinaka, Toshiki Saitoh, Jun Kawahara, Takanori Maehara, Norihito Yasuda, Masakazu Ishihata, Shuhei Denzumi, Yuma Inoue, Kunihiro Wasa, Hao Sun, Hana Itoh, Kazuhiro Kurita, Fumito Takeuchi, Yu Nakahata, Kazuaki Yamazaki, and Kosuke Shirai. My special thanks are due to Masakazu Ishihata for during my work with detailed discussion and advice.

My deep appreciation goes to all the members in Large-Scale Knowledge Processing Laboratory. I would be grateful to secretaries Sachiko Soma, Yukie Watanabe, and Yu Manabe for them invaluable support.

My another special thanks are due to the movie 『フカシギの数え方』 (https://www.youtube.com/watch?v=Q4gTV4r0zRs) produced by Miraikan and JST ERATO Minato Discrete Structure Manipulation System Project. The movie motivated me to become the master course student supervised by Professor Minato and obtain a doctoral degree. Professor Minato said this was the first case.

Finally, I am indebted to my parents for being patient and having understanding during my long student life. I definitely could not finish my degree and my thesis without a lot of their efforts.

Bibliography

[1] Satabdi Aditya, Bhaskar DasGupta, and Marek Karpinski. Algorithmic perspectives of network transitive reduction problems and their applications to synthesis and analysis of biological networks. *Biology*, 3(1):1–21, 2014.

[2] RÃl'ka Albert, Bhaskar DasGupta, Riccardo Dondi, Eduardo Sontag Sema Kachalo, Alexander Zelikovsky, and Kelly Westbrooks. A novel method for signal transduction network inference from indirect experimental evidence. *J Comput Biol*, 14(7):927–949, Sep 2007.

[3] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. Force: A fast and easy-to-implement variable-ordering heuristic. In *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, GLSVLSI '03, pages 116–119, 2003.

[4] Cosimo Federico Ardito, Donato Di Paola, and Andrea Gasparri. Decentralized estimation of the minimum strongly connected subdigraph for robotic networks with limited field of view. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 5304–5309, Dec 2012.

[5] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.

[6] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Implementing an efficient fptas for the 0-1 multi-objective knapsack problem. *European Journal of Operational Research*, 198(1):47 – 56, 2009.

[7] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers & OR*, 36(1):260–279, 2009.

[8] Jordan Bell and Brett Stevens. A survey of known results and research areas for n-queens. *Discrete Mathematics*, 309(1):1 – 31, 2009.

[9] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Discrete optimization with decision diagrams. *INFORMS J. on Computing*, 28(1):47–66, February 2016.

[10] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John N. Hooker. Variable ordering for the application of bdds to the maximum independent set problem. In *Proceedings of the 9th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR'12, pages 34–49, 2012.

[11] Beate Bollig and Ingo Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Trans. Comput.*, 45(9):993–1002, September 1996.

[12] Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Leonid Khachiyan. Enumerating minimal dicuts and strongly connected subgraphs and related geometric problems. In *Integer Programming and Combinatorial Optimization, 10th International IPCO Conference, New York, NY, USA, June 7-11, 2004, Proceedings*, pages 152–162, 2004.

[13] Jason Brown and Xiaohu Li. The strongly connected reliability of complete digraphs. *Networks*, 45:165–168, 05 2005.

[14] Aiden A. Bruen and Robert D. Dixon. The n-queens problem. *Discrete Mathematics*, 12(4):393–395, 1975.

[15] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput*, 35(8):677–691, August 1986.

[16] Vira Chankong and Yacov Y. Haimes. *Multiobjective decision making: Theory and Methodology*. Elsevier Science Ltd., 1983.

[17] Olivier Coudert. Solving graph optimization problems with zbdds. In *European Design and Test Conference, ED&TC '97, Paris, France, 17-20 March 1997*, pages 224–228, 1997.

[18] Captivo M. Eugénia, Clìmaco João, Figueira José, Martins Ernesto, and Santos José Luis. Solving bicriteria 0-1 knapsack problems using a labeling algorithm. *Comput. Oper. Res.*, 30(12):1865–1886, October 2003.

[19] James D. Fix and Brandon McPhail. Offline 1-minesweeper is np-complete. http://www.minesweeper.info/articles, 2004.

[20] Greg N. Frederickson and Joseph JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981.

[21] Xavier Gandibleux and Arnaud Fréville. Tabu search based procedure for solving the 0-1 multiobjective knapsack problem: The two objectives case. *J. Heuristics*, 6(3):361–383, 2000.

[22] Shahar Golan. Minesweeper on graphs. *Applied Mathematics and Computation*, 217:6616–6623, 2011.

[23] Frank K. Hwang, Paul E. Wright, and Xiaodong Hu. Exact reliabilities of most reliable double-loop networks. *Networks*, 30:81–90, 1997.

[24] Hiroshi Imai, Kyoko Sekine, and Keiko Imai. Computational investigations of all-terminal network reliability via bdds. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 82(5):714–721, May 1999.

[25] Akihiro Inokuchi, Hiroaki Ikuta, and Takashi Washio. Efficient graph sequence mining using reverse search. *IEICE Transactions on Information and Systems*, E95.D(7):1947–1958, 2012.

[26] Takeru Inoue, Keiji Takano, Takayuki Watanabe, Jun Kawahara, Ryo Yoshinaka, Akihiro Kishimoto, Koji Tsuda, Shin-ichi Minato, and Yasuhiro Hayashi. Distribution loss minimization with guaranteed error bound. *IEEE Transactions on Smart Grid*, 5(1):102–111, Jan 2014.

[27] Yuma Inoue and Shin-ichi Minato. An efficient method for indexing all topological orders of a directed graph. In *Algorithms and Computation*, pages 103–114, 2014.

[28] Yuma Inoue and Shin-ichi Minato. Acceleration of ZDD construction for subgraph enumeration via path-width optimization. TCS-TR-A-16-80. Hokkaido University, 2016.

[29] Hiroaki Iwashita. TdZdd. https://github.com/kunisura/TdZdd, 2014.

[30] Andrzej Jaszkiewicz. On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment. *IEEE Trans. Evolutionary Computation*, 6(4):402–412, 2002.

[31] Timothy Kam, Tiziano Villa, and Robert K. Brayton. Multi-valued decision diagrams: Theory and applications. *Multiple-Valued Logic*, 4(1-2):9–62, Jan 1998.

[32] David R. Karger. A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 11–17, 1995.

[33] Jun Kawahara, Takashi Horiyama, Keisuke Hotta, and Shin-ichi Minato. Generating all patterns of graph partitions within a disparity bound. In *WALCOM: Algorithms and Computation*, pages 119–131, 2017.

[34] Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Trans. Fundamentals*, E100-A(9):1773–1784, 2017.

[35] Jun Kawahara, Toshiki Saitoh, Hirofumi Suzuki, and Ryo Yoshinaka. Solving the longest oneway-ticket problem and enumerating letter graphs by augmenting the two representative approaches with zdds. In *Computational Intelligence in Information Systems*, pages 294–305, 2017.

[36] Richard Kaye. *Minesweeper is NP-complete*, volume 22 of *The Mathematical Intelligencer*, pages 9–15. 2000.

[37] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

[38] Shuji Kijima, Masashi Kiyomi, Yoshio Okamoto, and Takeaki Uno. On listing, sampling, and counting the chordal graphs with edge constraints. *Theoretical Computer Science*, 411(26):2591 – 2601, 2010.

[39] Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Inf. Process. Lett.*, 42(6):345–350, July 1992.

[40] Donald E. Knuth. *The art of computer programming: Bitwise tricks & techniques; binary decision diagrams, volume 4, fascicle 1*. 2009.

[41] Rhyd Lewis. Metaheuristics can solve sudoku puzzles. *Journal of Heuristics*, 13(4):387–401, Aug 2007.

[42] Elsa Loekito, James Bailey, and Jian Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowledge and Information Systems*, 24(2):235–268, Aug 2010.

[43] Inês Lynce and Joël Ouaknine. Sudoku as a SAT problem. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2006*, January 2006.

[44] Ehrgott Matthias and Gandibleux Xavier. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR-Spektrum*, 22(4):425–460, 2000.

[45] Peter Maurer. Generating strongly connected random graphs. In *Proceedings of the 2017 International Conference on Modeling, Simulation and Visualization Methods MSV'17*, pages 3–6, 2017.

[46] Shin-ichi Minato. Minimum-width method of variable ordering for binary decision diagrams. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, E75(A3):392–399, 1992.

[47] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *the 30th Design Automation Conference*, pages 272–277, 1993.

[48] Shin-ichi Minato. πdd: A new decision diagram for efficient problem solving in permutation space. In *Theory and Applications of Satisfiability Testing - SAT 2011*, pages 90–104, 2011.

[49] Preslav Nakov and Zile Wei. MINESWEEPER, #MINESWEEPER. http://www.minesweeper.info/articles, 2003.

[50] Jae-Hyun Park. All-terminal reliability analysis of wireless networks of redundant radio modules. *IEEE Internet of Things Journal*, 3(2):219–230, April 2016.

[51] Götz Pfeiffer. Counting transitive relations. *Journal of Integer Sequences*, 7(04.3.2.), August 2004.

[52] George Pólya and Ronald C. H. Read. *Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds*. 1987.

[53] Meir J. Rosenblatt and Zilla Sinuany-Stern. Generating the discrete efficient frontier to the capital budgeting problem. *Operations Research*, 37(3):384–394, 1989.

[54] Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the tutte polynomial of a graph of moderate size. In *Algorithms and Computations*, pages 224–233, 1995.

[55] Hirofumi Suzuki, Sun Hao, and Shin-ichi Minato. Generating all solutions of minesweeper problem using degree constrained subgraph model. In *the 24th Int ' l. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2016)*, pages 356–362. CSREA Press, July 2016.

[56] Hirofumi Suzuki, Masakazu Ishihata, and Shin-ichi Minato. Exact computation of strongly connected reliability by binary decision diagrams. In *Combinatorial Optimization and Applications, COCOA 2018, Lecture Notes in Computer Science*, volume 11346, pages 356–362. Springer, December 2018.

[57] Hirofumi Suzuki and Shin-ichi Minato. Fast enumeration of all pareto-optimal solutions for 0-1 multi-objective knapsack problems using zdds. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E101-A(9):1375–1382, September 2018.

[58] Jacques Teghem. Multi-objective combinatorial optimization. *In: Floudas C., Pardalos P. (eds) Encyclopedia of Optimization*, pages 2437–2442, 2009.

[59] Junn-Yuan Teng and Gwo-Hshiung Tzeng. A multiobjective programming approach for selecting non-independent transportation investment alternatives. *Transportation Research Part B: Methodological*, 30(4):291 – 307, 1996.

[60] William T. Tutte. Graph-polynomials. *Advances in Applied Mathematics*, 32(1):5 – 9, 2004. Special Issue on the Tutte Polynomial.

[61] Takeaki Uno. An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica*, 56(1):3–16, Jan 2010.

[62] Takeaki Uno and Hiroki Arimura. Ambiguous frequent itemset mining and polynomial delay enumeration. In *Advances in Knowledge Discovery and Data Mining*, pages 357–368, 2008.

[63] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8:410–421, 1979.

[64] Dalessandro Soares Vianna and José Elias Claudio Arroyo. A GRASP algorithm for the multi-objective knapsack problem. In *XXIV International Conference of the Chilean Computer Science Society (SCCC 2004)*, pages 69–75, November.

[65] Dubois Vincent and Bothorel Cecile. Transitive reduction for social network analysis and visualization. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '05, pages 128–131, 2005.

[66] Jin-Myung Won and Fakhri Karray. Cumulative update of all-terminal reliability for faster feasibility decision. *IEEE Transactions on Reliability*, 59(3):551–562, Sept 2010.

[67] Ryo Yoshinaka, Jun Kawahara, Shuhei Denzumi, Hiroki Arimura, and Shin ichi Minato. Counterexamples to the long-standing conjecture on the complexity of bdd binary operations. *Inf. Process. Lett.*, 112:636–640, 2012.

[68] Ryo Yoshinaka, Toshiki Saitoh, Jun Kawahara, Koji Tsuruma, Hiroaki Iwashita, and Shin-ichi Minato. Finding all solutions and instances of numberlink and slitherlink by zdds. *Algorithms*, 5(2):176–213, 2012.

[69] Liang Zhao, Hiroshi Nagamochi, and Toshihide Ibaraki. A linear time 5/3-approximation for the minimum strongly-connected spanning subgraph problem. *Inf. Process. Lett.*, 86(2):63–70, 2003.

[70] Zuse-Institute Berlin (ZIB). SNDlib. http://sndlib.zib.de/home.action, 2006.