Enumerating Graph Partitions Without Too Small Connected Components Using Zero-suppressed Binary and Ternary Decision Diagrams

Yu Nakahata
Nara Institute of Science and Technology, Ikoma, Japan
nakahata.yu.nm2@is.naist.jp

Jun Kawahara
Nara Institute of Science and Technology, Ikoma, Japan
jkawahara@is.naist.jp

Shoji Kasahara
Nara Institute of Science and Technology, Ikoma, Japan
kasahara@is.naist.jp

Abstract

Partitioning a graph into balanced components is important for several applications. For multi-objective problems, it is useful not only to find one solution but also to enumerate all the solutions with good values of objectives. However, there are a vast number of graph partitions in a graph, and thus it is difficult to enumerate desired graph partitions efficiently. In this paper, an algorithm to enumerate all the graph partitions such that all the weights of the connected components are at least a specified value is proposed. To deal with a large search space, we use zero-suppressed binary decision diagrams (ZDDs) to represent sets of graph partitions and we design a new algorithm based on frontier-based search, which is a framework to directly construct a ZDD. Our algorithm utilizes not only ZDDs but also ternary decision diagrams (TDDs) and realizes an operation which seems difficult to be designed only by ZDDs. Experimental results show that the proposed algorithm runs up to tens of times faster than an existing state-of-the-art algorithm.

1 Introduction

Partitioning a graph is a fundamental problem in computer science and has several important applications such as evacuation planning, political redistricting, VLSI design, and so on. In some applications among them, it is often required to balance the weights of connected components in a partition. For example, the task of the evacuation planning is to design which evacuation shelter inhabitants escape to. This problem is formulated as a graph partitioning problem, and it is important to obtain a graph partition consisting of balanced connected components (each of which contains a shelter and satisfies some conditions).

Another example is political redistricting, the purpose of which is to divide a region (such as a prefecture) into several balanced political districts for fairness.

There are a vast number of studies for graph optimization problems. An approach is to use a *zero-suppressed binary decision diagram* (ZDD) [12], which has originally been proposed as a compressed representation of a family of sets. A distinguished characteristic of the approach is not only to compute the single optimal solution but also to *enumerate* all the feasible solutions in the form of a ZDD. In addition, using several queries for a family of sets provided by ZDDs, we can impose various constraint conditions on solutions represented by a ZDD. Using this approach, Inoue et al. [4] designed an algorithm that constructs the ZDD representing the set of rooted spanning forests and utilized it to minimize the loss of electricity in an electrical distribution network under complex conditions, e.g., voltage, electric current and phase. There are other applications such as solving a variant of the longest path problem [8], reliability evaluation [2, 3], some puzzle problems [16], and exact calculation of impact diffusion in Web [11].

For balanced graph partitioning, Kawahara et al. [6] proposed an algorithm to construct a ZDD representing the set of balanced graph partitions by frontier-based search [7, 10, 14], which is a framework to directly construct a ZDD, and applied it to political redistricting. However, their method stores the weights of connected components, represented as integers, into the ZDD, which generates a not compressed ZDD. As a result, the computation is tractable only for graphs only with less than 100 vertices. Nakahata et al. [13] proposed an algorithm to construct the ZDD representing the set of partitions such that all the weights of connected components are bounded by a given upper threshold (and applied it to evacuation planning). Their approach enumerates connected components with weight more than the upper threshold as a ZDD, say *forbidden components*, and constructs a ZDD representing partitions not containing any forbidden component *as a subgraph* by set operations, which are performed by so-called apply-like methods [1]. However, it seems difficult to directly use their method to obtain balanced partitions by letting connected components with weight less than a lower threshold be forbidden components because partitions not containing any forbidden component *as a connected component* (i.e., one of parts in a partition coincides a forbidden component) cannot be obtained by apply-like methods.

In this paper, for a ZDD $Z_{\mathcal{A}}$ and an integer L, we propose a novel algorithm to construct the ZDD representing the set of graph partitions such that the partitions are represented by $Z_{\mathcal{A}}$ and all the weights of the connected components in the partitions are at least L. The input ZDD $Z_{\mathcal{A}}$ can be the sets of spanning forests used for evacuation planning (e.g., [13]), rooted spanning forests used for power distribution networks (e.g., [4]), and simply connected components representing regions (e.g., [6]), all of which satisfy complex conditions according to problems. We generically call these structures "partitions." Roughly speaking, our algorithm excludes partitions containing any forbidden component as a connected component from $Z_{\mathcal{A}}$. We first construct the ZDD, say $Z_{\mathcal{S}}$, representing the set of forbidden components, each of which has weight less than L. Then, for a component in $Z_{\mathcal{S}}$, we consider the cutset that separates the input graph into the component and the rest. We represent the set of pairs of every component in $Z_{\mathcal{S}}$ and its cutset as a *ternary decision diagram* (TDD) [15], say $T_{\mathcal{S}\pm}$. We propose a method to construct the TDD $T_{\mathcal{S}\pm}$ from $Z_{\mathcal{S}}$ by frontier-based search. By using the TDD $T_{\mathcal{S}\pm}$, we show how to obtain partitions each of which belongs to $Z_{\mathcal{A}}$, contains all the edges in a component of a pair in $T_{\mathcal{S}\pm}$ and contains no edge in the cutset of the pair. Finally, we exclude such partitions from $Z_{\mathcal{A}}$ and obtain the desired partitions. By numerical experiments, we show that the proposed algorithm runs up to tens of times faster than an existing state-of-the-art algorithm.
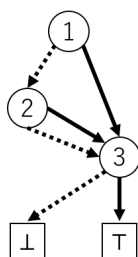
Figure 1 The ZDD representing the family $\{\{1,3\},\{2,3\},\{3\}\}$. A square represents a terminal node. A circle is a non-terminal node and the number in it is a label. A solid arc is a 1-arc and a dashed arc is a 0-arc.
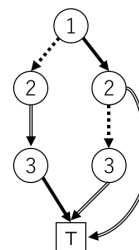
Figure 2 The TDD representing the signed family $\{\{+1,-2\},\{+1,-3\},\{-2,+3\}\}$. A dashed arc is a ZERO-arc, a solid single arc is a POS-arc and a solid double arc is NEG-arc. For simplicity, $\bot$ and the arcs pointing at it are omitted.

This paper is organized as follows. In Sec. 2, we give some preliminaries and explain ZDDs, TDDs, and frontier-based search. We describe an overview of our algorithm in Sec. 3.1, and the detail in the rest of Sec. 3. Section 4 gives experimental results. The conclusion is described in Sec. 5.

2   Preliminaries

2.1   Notation

Let $\mathbb{Z}^+$ be the set of positive integers. For $k \in \mathbb{Z}^+$, we define $[k] = \{1, 2, \ldots, k\}$. In this paper, we deal with a vertex-weighted undirected graph $G = (V, E, p)$, where $V = [n]$ is the vertex set and $E = \{e_1, e_2, \ldots, e_m\} \subseteq \{\{u, v\} \mid u, v \in V\}$ is the edge set. The function $p \colon V \to \mathbb{Z}^+$ gives the weights of the vertices. We often drop p from (V, E, p) when there is no ambiguity. For an edge set $E' \subseteq E$, we call the subgraph (V, E') a *graph partition*. We often identify the edge set E' with the partition (V, E') by fixing the graph G. For edge sets E', E'' with $E'' \subseteq E' \subseteq E$ and a vertex set $V'' \subseteq V$, we say that (V'', E'') is included in the partition (V, E') *as a subgraph*. The subgraph (V'', E'') is called a *connected component* in the partition (V, E') if $V'' = \mathrm{dom}(E'')$ holds, there is no edge in $E' \setminus E''$ incident with a vertex in V'', and for any two distinct vertices $u, v \in V''$, there is a u-v path on (V'', E''), where $\mathrm{dom}(E'')$ is the set of vertices which are endpoints of at least one edge in E''. In this case, we say that (V'', E'') is included in the partition (V, E') *as a connected component*. We denote the neighborhood of a vertex v in a partition $E' \subseteq E$ by $N(E', v) = \{u \mid \{u, v\} \in E'\}$. For $i \in [m]$, $E^{\leq i}$ denotes the set of edges whose indices are at most i. We define $E^{<i}$, $E^{\geq i}$ and $E^{>i}$ in the same way.

For a set U, let $U^+ = \{+e \mid e \in U\}, U^- = \{-e \mid e \in U\}$ and $U^\pm = U^+ \cup U^-$. A *signed set* is a subset of $U^\pm$ such that, for all $e \in U$, the set contains at most one of $+e$ and $-e$. For example, when $U = [3]$, both $\{+1, -2\}$ and $\{-3\}$ are signed sets but $\{+1, -1, +3\}$ is not. A *signed family* is a family of signed sets. In particular, when $U = E$, we sometimes call a signed set a *signed subgraph* and call a signed family a *set of signed subgraphs*. For a signed set $S^\pm$, we define $\mathrm{abs}(S^\pm) = \{e \mid (+e \in S^\pm) \vee (-e \in S^\pm)\}$.

2.2   Zero-suppressed binary decision diagram

A *zero-suppressed binary decision diagram* (ZDD) [12] is a directed acyclic graph $Z = (N_Z, A_Z)$ representing a family of sets. Here N_Z is the set of *nodes* and A_Z is the set of *arcs*.[1] N_Z contains two *terminal nodes* $\top$ and $\bot$. The other nodes than the terminal nodes are called *non-terminal nodes*. Each non-terminal node α has the *0-arc*, the *1-arc*, and the *label* corresponding to an item in the universe set. For $x \in \{0, 1\}$, we call the destination of the x-arc of a non-terminal node α the *x-child* of α. We denote the label of α by $l(\alpha)$ and in this paper, assume that $l(\alpha) \in \mathbb{Z}^+ \cup \{\infty\}$ for any $\alpha \in N_Z$. For convenience, we let $l(\top) = l(\bot) = \infty$. For each directed arc $(\alpha, \beta) \in A_Z$, the inequality $l(\alpha) < l(\beta)$ holds, which ensures that Z is acyclic. There is exactly one node whose in-degree is zero, called the *root node* and denoted by r_Z. The number of the non-terminal nodes of Z is called the *size* of Z and denoted by $|Z|$.

Z represents the family of sets in the following way. Let $\mathcal{P}_Z$ be the set of all the directed paths from r_Z to $\top$. For a directed path $p = (n_1, a_1, n_2, a_2, \ldots, n_k, a_k, \top) \in \mathcal{P}_Z$ with $n_i \in N_Z$, $a_i \in A_Z$ and $n_1 = r_Z$, we define $S_p = \{l(n_i) \mid a_i \in A_{Z,1}, i \in [k]\}$, where $A_{Z,1}$ is the set of the 1-arcs of Z. We interpret that Z represents the family $\{S_p \mid p \in \mathcal{P}_Z\}$. In other words, a directed path from r_Z to $\top$ corresponds to a set in the family represented by Z. As an example, we illustrate the ZDD representing the family $\{\{1, 3\}, \{2, 3\}, \{3\}\}$ in Fig. 1. In the figure, a dashed arc ($\dashrightarrow$) and a solid arc ($\rightarrow$) are a 0-arc and a 1-arc, respectively. On the ZDD in Fig. 1, there are three directed paths from the root node to $\top$: $1 \rightarrow 3 \rightarrow \top$, $1 \dashrightarrow 2 \rightarrow 3 \rightarrow \top$, and $1 \dashrightarrow 2 \dashrightarrow 3 \rightarrow \top$, which correspond to $\{1, 3\}, \{2, 3\}$, and $\{3\}$, respectively. We denote a ZDD representing a family $\mathcal{F}$ by $Z_\mathcal{F}$.

2.3   Ternary decision diagram

A *ternary decision diagram* (TDD) [15] is a directed acyclic graph $T = (N_T, A_T)$ representing a signed family. A TDD shares many concepts with a ZDD, and thus we use the same notation as a ZDD for a TDD. The difference between a ZDD and a TDD is that, while a node of the former has two arcs, that of the latter has three, which are called the *ZERO-arc*, the *POS-arc*, and the *NEG-arc*.

T represents the signed family in the following way. For a directed path $p = (n_1, a_1, n_2, a_2, \ldots, n_k, a_k, \top) \in \mathcal{P}_T$ with $n_i \in N_Z$, $a_i \in A_T$ and $n_1 = r_T$, we define $S_p^\pm = \{+l(n_i) \mid a_i \in A_{T,+}, i \in [k]\} \cup \{-l(n_i) \mid a_i \in A_{T,-}, i \in [k]\}$, where $A_{T,+}$ and $A_{T,-}$ are the set of the POS-arcs of T and the set of the NEG-arcs of T, respectively. We interpret that T represents the signed family $\{S_p^\pm \mid p \in \mathcal{P}_T\}$. We illustrate the TDD representing the signed family $\{\{+1, -2\}, \{+1, -3\}, \{-2, +3\}\}$ in Fig. 2 for example. In the figure, a dashed arc ($\dashrightarrow$), a solid single arc ($\rightarrow$), and a solid double arc ($\Rightarrow$) are a ZERO-arc, a POS-arc, and a NEG-arc, respectively. The TDD in the figure has three directed paths from the root node to $\top$: $1 \rightarrow 2 \Rightarrow \top$, $1 \rightarrow 2 \dashrightarrow 3 \Rightarrow \top$, and $1 \dashrightarrow 2 \Rightarrow 3 \rightarrow \top$, which correspond to $\{+1, -2\}, \{+1, -3\}$, and $\{-2, +3\}$, respectively.

2.4   Frontier-based search

Frontier-based search [7, 10, 14] is a framework of algorithms that efficiently construct a decision diagram representing the set of subgraphs satisfying given constraints of an input graph. We explain the general framework of frontier-based search. Given a graph G, let

---

[1] To avoid confusion, we use the words "vertex" and "edge" for input graphs and "nodes" and "arcs" for decision diagrams.

$\mathcal{M}$ be a class of subgraphs we would like to enumerate (for example, $\mathcal{M}$ is the set of all the s-t paths on G). Frontier-based search constructs the ZDD representing the family $\mathcal{M}$ of subgraphs. By fixing G, a subgraph is identified with the edge set the subgraph has, and thus the ZDD represents the family of edge sets actually. Non-terminal nodes of ZDDs constructed by frontier-based search have labels $e_1, \dots, e_m$. We identify e_i with the integer i. We assume that it is determined in advance which edge in G has which index i of e_i.

We directly construct the ZDD in a breadth-first manner. We first create the root node of the ZDD, make it have label e_1, and then we carry out the following procedure for $i = 1, \dots, m$. For each node n_i with label e_i, we create two nodes, each of which is either a terminal node or a non-terminal node whose label is e_{i+1} (if $i = m$, the candidate is only a terminal node), as the 0-child and the 1-child of n_i.

Which node the x-arc of a node n_i with label e_i points at is determined by a function, called MAKENEWNODE, of which we design the detail according to $\mathcal{M}$, i.e., what subgraphs we want to enumerate. Here we describe the generalized nature that MAKENEWNODE must possess. The node n_i represents the set of the subgraphs, denoted by $\mathcal{G}(n_i)$, corresponding to the set of the directed paths from the root node to n_i. Each subgraph in $\mathcal{G}(n_i)$ contains only edges in $E^{<i}$. Note that $\mathcal{G}(\top)$ is the desired set of subgraphs represented by the ZDD after the construction finishes. To decide which node the x-arc of n_i points at without traversing the ZDD (under construction), we make each node n_i have the information $n_i.\texttt{conf}$, which is shared by all the subgraphs in $\mathcal{G}(n_i)$. The content of $n_i.\texttt{conf}$ also depends on $\mathcal{M}$ (for example, in the case of s-t paths, we store degrees and components of the subgraphs in $\mathcal{G}(n_i)$ into $n_i.\texttt{conf}$). MAKENEWNODE creates a new node, say n_{new}, with label e_{i+1} and must behave in the following manner.

1. For all edge sets $S^{\leq i} \in \mathcal{G}(n_{\text{new}})$, if there is no edge set $S^{>i} \subseteq E^{>i}$ such that $S^{\leq i} \cup S^{>i} \in \mathcal{M}$, the function discards n_{new} and returns $\bot$ to avoid redundant expansion of nodes. (*pruning*)

2. Otherwise, if $i = m$, the function returns $\top$.

3. Otherwise, the function calculates $n_{\text{new}}.\texttt{conf}$ from $n_i.\texttt{conf}$. If there is a node n_{i+1} such that whose label is e_{i+1} and $n_{\text{new}}.\texttt{conf} = n_{i+1}.\texttt{conf}$, the function abandons n_{new} and returns n_{i+1}. (*node merging*) If not, the function returns n_{new}.

We make the x-arc of n_i point at the node returned by MAKENEWNODE.

As for $n_i.\texttt{conf}$, in the case of several kinds of subgraphs such as paths and cycles, it is known that we only have to store states relating to the vertices to which both an edge in $E^{<i}$ and an edge in $E^{\geq i}$ are incident into each node [10] (in the case of s-t paths, we store degrees and components of such vertices into each node). The set of the vertices are called the *frontier*. More precisely, the i-th frontier is defined as $F_i = (\bigcup_{j=1}^{i-1} e_j) \cap (\bigcup_{k=i}^{m} e_k)$. For convenience, we define $F_0 = F_m = \emptyset$. States of vertices in F_{i-1} are stored into $n_i.\texttt{conf}$. By limiting the domain of the information to the frontier, we can reduce memory consumption and share more nodes, which leads to a more efficient algorithm.

The efficiency of an algorithm based on frontier-based search is often evaluated by the *width of a ZDD* constructed by the algorithm. The width W_Z of a ZDD Z is defined as $W_Z = \max\{|\mathcal{N}_i| \mid i \in [m]\}$, where $\mathcal{N}_i$ denotes the set of nodes whose labels are e_i. Using W_Z, the number of nodes in Z can be written as $|Z| = \mathcal{O}(m W_Z)$ and the time complexity of the algorithm is $\mathcal{O}(\tau |Z|)$, where τ denotes the time complexity of MAKENEWNODE for one node.

3   Algorithms

3.1   Overview of the proposed algorithms

In this section, for a ZDD $Z_{\mathcal{A}}$ and $L \in \mathbb{Z}^+$, we propose a novel algorithm to construct the ZDD representing the set of graph partitions such that the partitions are represented by $Z_{\mathcal{A}}$ and each connected component in the partitions has weight at least L. In general, there are two techniques to obtain ZDDs having desired conditions. One is frontier-based search, described in the previous section. The method proposed by Kawahara et al. [6] directly stores the weight of each component into ZDD nodes (as `conf`) and prunes a node when it is determined that the weight of a component is less than L. However, for two nodes, if the weight of a single component on the one node differs from that on the other node, the two nodes cannot be merged. Consequently, node merging rarely occurs in Kawahara et al.'s method and thus the size of the resulting ZDD is too large to construct it if the input graph has more than 100 vertices.

The other technique is the usage of the recursive structure of a ZDD. Methods based on the recursive structure are called *apply-like* methods [1]. For each node α of a ZDD, the nodes and arcs reachable from α compose another ZDD, whose root is α. For a ZDD Z and $x \in \{0, 1\}$, let $c_x(Z)$ be the ZDD composed by the nodes and arcs reachable from the x-child of the root. For (one or more) ZDDs F (and G), an apply-like method constructs a target ZDD by recursively calling itself against $c_0(F)$ and $c_1(F)$ (and $c_0(G)$ and $c_1(G)$). For example, the ZDD representing $F \cap G$ can be computed from $c_0(F) \cap c_0(G)$ and $c_1(F) \cap c_1(G)$. Apply-like methods support various set operations [1, 10].

Nakahata et al. [13] developed an algorithm to upperbound the weights of connected components in each partition, i.e., to construct the ZDD representing the set $\mathcal{A}$ of partitions included in a given ZDD and the weights of all the components in the partitions are at most $H \in \mathbb{Z}^+$. Their algorithm first constructs the ZDD $Z_{\mathcal{S}}$ representing the set of forbidden components (described in the introduction) with weight more than H by frontier-based search. Then, the algorithm constructs the ZDD representing $\{A \in \mathcal{A} \mid \exists S \in \mathcal{S}, A \supseteq S\}$, written as $Z_{\mathcal{A}}.\mathrm{restrict}(Z_{\mathcal{S}})$, which means the set of all the partitions each of which includes a component in $\mathcal{S}$ as a subgraph, in a way of apply-like methods. Finally, we extract subgraphs not in $Z_{\mathcal{A}}.\mathrm{restrict}(Z_{\mathcal{S}})$ from $Z_{\mathcal{A}}$ by the set difference operation $Z_{\mathcal{A}} \setminus (Z_{\mathcal{A}}.\mathrm{restrict}(Z_{\mathcal{S}}))$ [12], which is also an apply-like method.

In our case, lowerbounding the weights of components, it is difficult to compute desired partitions by the above approach because a partition including a forbidden component (i.e., weight less than L) *as a subgraph* can be a feasible solution. We want to obtain a partition including a forbidden component *as a connected component*. Although we can perform various set operations by designing apply-like methods, it seems difficult to obtain such partitions by direct set operations.

Our idea in this paper is to employ the family of signed sets to represent the set of pairs of every forbidden component and its cutset. We use the following observation.

▶ **Observation 1.** *Let A be a graph partition of $G = (V, E)$ and $S \subseteq E$ be an edge set such that $(\mathrm{dom}(S), S)$ is connected. The partition A contains $(\mathrm{dom}(S), S)$ as a connected component if and only if both of the following hold.*

1. *A contains all the edges in S.*

2. *A does not contain any edge e in $E \setminus S$ such that e has at least one vertex in $\mathrm{dom}(S)$.*

Based on Observation 1, we associate a signed subgraph $S^{\pm}$ with a connected subgraph $(\mathrm{dom}(S), S)$:
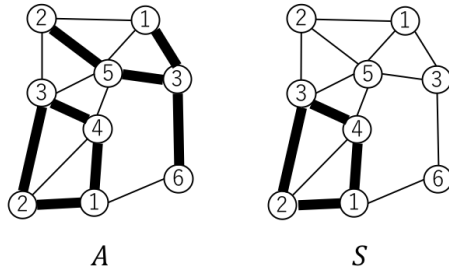
Figure 3 A graph partition A and a connected subgraph S. Bold lines are edges contained in the partition or the subgraph. Values in vertices are its weights. A contains S as a connected component. The weight of S is $1 + 3 = 4 < 5$, and thus, when $L = 5$, A does not satisfy the lower bound constraint.
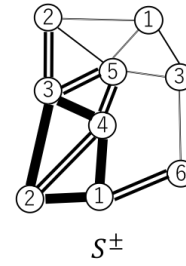
Figure 4 A signed subgraph $S^{\pm}$ with minimal cutset corresponding to S in Fig. 3. Thin single lines, bold single lines, and doubled lines are zero edges, positive edges, and negative edges, respectively.

$$
\begin{aligned}
S^{\pm} &= S^+ \cup S^-, & (1) \\
S^+ &= \{+e \mid e \in S\}, & (2) \\
S^- &= \{-e \mid (e \in E \setminus S) \wedge (e \cap \mathrm{dom}(S) \neq \emptyset)\}. & (3)
\end{aligned}
$$

$S^{\pm}$ is a signed subgraph such that $\mathrm{abs}(S^+)$ and $\mathrm{abs}(S^-)$ are sets of edges satisfying Conditions 1 and 2 in Observation 1, respectively. Note that $\mathrm{abs}(S^-)$ is a cutset of G, that is, removing the edges in $\mathrm{abs}(S^-)$ separates G into the connected component $(\mathrm{dom}(\mathrm{abs}(S^+)), \mathrm{abs}(S^+))$ and the rest. In addition, $\mathrm{abs}(S^-)$ is minimal among such cutsets. In this sense, we say that $S^{\pm}$ is a *signed subgraph with minimal cutset for S*.

Hereinafter, we call edges in $\mathrm{abs}(S^+)$ *positive edges*, $\mathrm{abs}(S^-)$ *negative edges* and the other edges *zero edges*. Figure 4 shows $S^{\pm}$ associated with S in Fig. 3. The partition A in Fig. 3 indeed contains all the edges in $\mathrm{abs}(S^+)$ and does not contain any edges in $\mathrm{abs}(S^-)$. For a graph partition $E' \subseteq E$, when the weights of all the connected components of E' is at least L, we say that E' *satisfies the lower bound constraint*. To extract partitions not satisfying the lower bound constraint from an input ZDD, we compute the set of partitions each of which has all the edges in $\mathrm{abs}(S^+)$ and no edge in $\mathrm{abs}(S^-)$ for some $S \in \mathcal{S}$.

The overview of the proposed method is as follows. In the following, let $\mathcal{A}$ be the set of graph partitions represented by the input ZDD and $\mathcal{B}$ be the set of graph partitions each of which belongs to $\mathcal{A}$ and satisfies the lower bound constraint.

1. We construct the ZDD $Z_{\mathcal{S}}$ representing the set $\mathcal{S}$ of forbidden components, where $\mathcal{S}$ is the set of the connected components of G whose weights are less than L.

2. Using $Z_{\mathcal{S}}$, we construct the TDD $T_{\mathcal{S}^{\pm}}$, where $\mathcal{S}^{\pm}$ is a set of signed subgraphs with minimal cutset corresponding to $\mathcal{S}$ by a way of frontier-based search.

3. Using $T_{\mathcal{S}^{\pm}}$, we construct the ZDD $Z_{\mathcal{S}^{\uparrow}}$, where $\mathcal{S}^{\uparrow}$ is the set of partitions each of which contains at least one forbidden component in $\mathcal{S}$ as a connected component.

4. We obtain the ZDD $Z_{\mathcal{B}}$ by the set difference operation $Z_{\mathcal{A}} \setminus Z_{\mathcal{S}^{\uparrow}}$ [12].

In the rest of this section, we describe each step from 1 to 3.

3.2   Constructing $Z_{\mathcal{S}}$

We describe how to construct $Z_{\mathcal{S}}$, which represents the set $\mathcal{S}$ of forbidden subgraphs whose weights are less than L. In this subsection, we consider only forbidden components with at least one edge. Note that a component with only one vertex cannot be distinguished by sets of edges because all such subgraphs are represented by the empty edge set. We show how to deal with components having only one vertex in Sec. 3.4.

We can construct $Z_{\mathcal{S}}$ using frontier-based search. Due to the page restriction, we describe a brief overview. To construct $Z_{\mathcal{S}}$, in the frontier-based search, it suffices to ensure that every enumerated subgraph has only one connected component and its weight is less than L. The former can be dealt by storing the connectivity of the vertices in the frontier as `comp` [7]. The latter can be checked by managing the total weight of vertices such that at least one edge is incident to as `weight`.

Let us analyze the width of $Z_{\mathcal{S}}$. For nodes with the same label, there are $\mathcal{O}(B_f)$ different states for `comp` [6], where, for $k \in \mathbb{Z}^+$, B_k is the k-th Bell number and $f = \max\{|F_i| \mid i \in [m]\}$. As for `weight`, when `weight` exceeds L, we can immediately conclude that the subgraphs whose weights are less than L are generated no more. If we prune such cases, there are $\mathcal{O}(L)$ different states for `weight`. As a result, we can obtain the following lemma on the width of $Z_{\mathcal{S}}$.

▶ **Lemma 2.** *The width of $Z_{\mathcal{S}}$ is $\mathcal{O}(B_f L)$, where $f = \max\{|F_i| \mid i \in [m]\}$.*

3.3   Constructing $T_{\mathcal{S}^{\pm}}$

In this subsection, we propose an algorithm to construct $T_{\mathcal{S}^{\pm}}$. First, we show how to construct the TDD representing the set of all the signed subgraphs with minimal cutset, including a disconnected one. Next, we describe the method to construct $T_{\mathcal{S}^{\pm}}$ using $Z_{\mathcal{S}}$.

Let $S^{\pm} = S^+ \cup S^-$ be a signed subgraph. Our algorithm uses the following observation on signed subgraphs with minimal cutset.

▶ **Observation 3.** *A signed subgraph $S^{\pm}$ is a signed subgraph with minimal cutset if and only if the following two conditions hold:*
1. *For all $v \in V$, at most one of a zero edge or a positive edge is incident to v.*
2. *For all the negative edges $\{u, v\}$, a positive edge is incident to at least one of u and v.*
Conditions 1 and 2 in Observation 3 ensure that $\mathrm{abs}(S^-)$ is a cutset such that removing it leaves the connected component whose edge set is $\mathrm{abs}(S^+)$ and the minimality of $\mathrm{abs}(S^-)$. This shows the correctness of the observation. We design an algorithm based on frontier-based search to construct a TDD representing the set of all the signed subgraphs satisfying Conditions 1 and 2 in Observation 3.

First, we consider Condition 1. To ensure Condition 1, we store an array `colors` : $V \to 2^{\{0,+,-\}}$ into each TDD node. For all $v \in F_{i-1}$, we manage $n_i.\texttt{colors}[v]$ so that it is equal to the set of types of edges incident to v. For example, if a zero edge and a positive edge are incident to v and no negative edges are, `colors`[v] must be $\{0, +\}$. We can prune the case such that Condition 1 is violated using `colors`, which ensures Condition 1.

Next, we consider Condition 2. Let $\{u, v\}$ be a negative edge. When u and v leave the frontier at the same time, we check if Condition 2 is satisfied from `colors`[u] and `colors`[v] and, if not, we prune the case. When one of u or v leaves the frontier (without loss of generality, we assume the vertex is u), if no positive edges are incident to u, at least one positive edge must be incident to v later. To deal with this situation, we store an array `reserved` : $V \to \{0, 1\}$ into each TDD node. For all $v \in F_{i-1}$, we manage `reserved`[v] so

that `reserved`$[v] = 1$ if and only if at least one positive edge must be incident to v later. We can prune the cases such that $v \in V$ is leaving the frontier and both `reserved`$[v] = 1$ and $+ \notin$ `colors`$[v]$ hold, which violate Condition 2. We show pseudocode of MakeNewNode function in Appendix of the full version.

We give the following lemma on the width of a ZDD constructed by the algorithm. We show a proof of the lemma in the full version.

▶ **Lemma 4.** *The width W_T of a ZDD constructed by the above algorithm is $W_T = \mathcal{O}(6^f)$.*

Next, we show how to construct $T_{\mathcal{S}^\pm}$ using $Z_{\mathcal{S}}$. We can achieve this goal using *subsetting* technique [5] with the above algorithm. Subsetting technique is a framework to construct a decision diagram corresponding to another decision diagram. We ensure that, for all $S^\pm = S^+ \cup S^- \in \mathcal{S}^\pm$, there exists $S \in \mathcal{S}$ such that $\mathrm{abs}(S^+) = S$ in the construction of $T_{\mathcal{S}^\pm}$ using subsetting technique.

3.4 Constructing $Z_{\mathcal{S}^\uparrow}$

In this section, we show how to construct $Z_{\mathcal{S}^\uparrow}$ and how to deal with forbidden components consisting only of one vertex whose weight is less than L, which was left as a problem in Sec. 3.2. From Observation 1 and Eqs. (1)–(3), $\mathcal{S}^\uparrow$ can be written as

$$\mathcal{S}^\uparrow = \{E' \subseteq E \mid \exists S^\pm \in \mathcal{S}^\pm, (\forall + e \in S^\pm, e \in E') \wedge (\forall - e \in S^\pm, e \notin E')\}. \tag{4}$$

Using $T_{\mathcal{S}^\pm}$, we can construct $Z_{\mathcal{S}}$ by the existing algorithm [9].

Finally, we show how to deal with a graph partition containing a single vertex v such that $p(v) < L$ as a connected component, i.e., a partition has an isolated vertex with small weight. Let $\mathcal{F}_v$ be the set of graph partitions containing $(\{v\}, \emptyset)$ as a connected component. A graph partition $E' \subseteq E$ belongs to $\mathcal{F}_v$ if and only if E' does not contain any edge incident to v. Using this, we can construct the ZDD Z_v representing $\mathcal{F}_v$ in $\mathcal{O}(m)$ time. For each $v \in V$ such that $p(v) < L$, we construct Z_v and update $Z_{\mathcal{S}^\uparrow} \leftarrow Z_{\mathcal{S}^\uparrow} \cup Z_v$. In this way, we can deal with all the graph partitions containing a connected component whose weight is less than L. We show an example of execution of the whole algorithm in Appendix of the full version.

4 Experimental results

We conducted computational experiments to evaluate the proposed algorithm and to compare it with the existing state-of-the-art algorithm of Kawahara et al [6]. We used a machine with an Intel Xeon Processor E5-2690v2 (3.00 GHz) CPU and a 64 GB memory (Oracle Linux 6) for the experiments. We have implemented the algorithms in C++ and compiled them by g++ with the `-O3` optimization option. In the implementation, we used the `TdZdd` library [5] and the `SAPPORO_BDD` library.[2] The timeout is set to be an hour.

We used graphs representing some prefectures in Japan for the input graphs. The vertices represent cities and there is an edge between two cities if and only if they have the common border. The weight of a vertex represents the number of residents living in the city represented by the vertex. As for the input ZDD $Z_{\mathcal{A}}$, we adopted three types of graph partitions: graph partitions such that each connected component is an induced subgraph [6], which we call *induced partition*, forests, and rooted forests. There is a one-to-one correspondence between

---

[2] Although the `SAPPORO_BDD` library is not released officially, you can see the code in `https://github.com/takemaru/graphillion/tree/master/src/SAPPOROBDD`.

■ **Table 1** Summary of input graphs and input graph partitions.

Name	n	m	k	Induced partition $|Z_\mathcal{A}|$	$|\mathcal{A}|$	Forest $|Z_\mathcal{A}|$	$|\mathcal{A}|$	Rooted forest $|Z_\mathcal{A}|$	$|\mathcal{A}|$
G_1 (Gumma)	37	80	4	10236	1.25×10^8	26361	1.01×10^{19}	8957	1.66×10^{16}
G_2 (Ibaraki)	44	95	7	17107	6.38×10^{13}	15553	6.14×10^{23}	3238	1.94×10^{19}
G_3 (Chiba)	60	134	14	301946	6.69×10^{22}	213773	4.86×10^{33}	15741	5.04×10^{25}
G_4 (Aichi)	69	173	17	1598213	9.26×10^{29}	879361	1.78×10^{42}	43465	3.10×10^{30}
G_5 (Nagano)	77	185	5	13203	2.77×10^{17}	44804	2.95×10^{43}	26476	7.66×10^{39}

induced partitions and partitions of the vertex set. A rooted forest is a forest such that each tree in the forest has exactly one specified vertex. We chose special vertices for each graph randomly. A summary of input graphs and input graph partitions is in Tab. 1. In the table, we show graph names and the prefecture represented by the graph, the number of vertices (n), edges (m) and connected components (k) in graph partitions. The groups of columns "Induced partition", "Forest", and "Rooted forest" indicate the types of input graph partitions. Inside each of them, we show the size (the number of non-terminal nodes) of $Z_\mathcal{A}$ and the cardinality of $\mathcal{A}$.

The lower bounds of weights are determined as follows. Let k be the number of connected components in a graph partition and r be the maximum ratio of the weights of two connected components in the graph partition. From k and r, we can derive the necessary condition that the weight of every connected component must be at least $L(k,r) = P/(r(k-1)+1)$, where $P = \sum_{v \in V} p(v)$ [6]. We used $L(k,r)$ as the lower bound of weights in the experiment. For each graph, we run the algorithms in $r = 1.1, 1.2, 1.3, 1.4,$ and 1.5.

We show the experimental results in Tab. 2. In the table, we show the graph name, the value of r and $L(k,r)$, and the execution time of *Alg. N*, the proposed algorithm, and *Alg. K*, the algorithm of Kawahara et al. The size of $Z_\mathcal{B}$ and the cardinality of $\mathcal{B}$ are also shown. "OOM" means *out of memory* and "-" means both algorithms failed to construct the ZDD (due to timeout or out of memory). We marked the values of the time of the algorithm which finished faster as bold.

First, we analyze the results for induced partitions. For the input graphs from G_1 to G_4, both Alg. N and Alg. K succeeded in constructing $Z_\mathcal{B}$, except when $r = 1.1$ in G_4 for Alg. K. In cases where both algorithms succeeded in constructing $Z_\mathcal{B}$, the time for Alg. N to construct the ZDD is 2–32 times shorter than that for Alg. K. In addition, Alg. N succeeded in constructing the ZDD when $r = 1.1$ in G_4, where Alg. K failed to construct the ZDD because of out of memory. These results show the efficiency of our algorithm. In contrast, for G_5, although both algorithms failed to construct the ZDD when $r = 1.1, 1.2, 1.3$ and 1.4, only Alg. K succeeded when $r = 1.5$. In this case, the size of the ZDD constructed by Alg. N did stay in the limitation of memory while, in our algorithm, the size of $Z_{\mathcal{S}^\uparrow}$ exceeded the limitation of memory.

Second, we investigate the results for forests. Both Alg. N and Alg. K succeeded in constructing $Z_\mathcal{B}$ for the input graph from G_1 to G_4. In all those cases, Alg. N was faster than Alg. K. Comparing the results with those of induced partitions, we found that the execution time of Alg. K depends on the input partitions more than Alg. N does. For example, for G_1, while the execution time of Alg. N is almost irrelevant to the types of input ZDDs, that of Alg. K differ up to about five times. This is because the efficiency of Alg. K strongly depends on the sizes of input ZDDs. This makes the sizes of output ZDDs constructed by Alg. K large, which implies the increase in the execution time of Alg. K. In contrast, the execution time of Alg. N does not depend on the sizes of input ZDDs in many cases because Alg. N

Table 2 Experimental results for three types of input graph partitions.

	r	$L(r,k)$	Induced partition				Forest				Rooted forest			
			Alg. N	Alg. K	$|Z_B|$	$|\mathcal{B}|$	Alg. N	Alg. K	$|Z_B|$	$|\mathcal{B}|$	Alg. N	Alg. K	$|Z_B|$	$|\mathcal{B}|$
G_1	1.1	458947	**4.22**	12.07	4912	1.74×10^4	**4.03**	50.84	29502	8.24×10^{12}	**3.95**	14.96	17920	3.52×10^{11}
	1.2	429016	**2.06**	10.50	3500	5.40×10^4	**2.04**	47.30	21364	3.10×10^{13}	**2.02**	13.34	6331	1.68×10^{12}
	1.3	402750	**1.15**	7.49	2986	9.02×10^4	**1.18**	36.10	18113	7.42×10^{13}	**1.17**	10.54	4655	4.44×10^{12}
	1.4	379514	**0.99**	5.72	3115	2.52×10^5	**1.03**	24.41	20605	3.84×10^{14}	**1.03**	6.97	7677	3.18×10^{13}
	1.5	358813	**0.90**	5.12	3562	2.99×10^5	**0.89**	23.29	20367	7.19×10^{14}	**0.88**	6.52	6719	6.17×10^{13}
G_2	1.1	383928	**3.70**	29.48	27927	1.91×10^6	**3.60**	35.28	47461	2.56×10^{13}	3.53	**2.19**	391	4.32×10^6
	1.2	355836	**3.03**	23.03	83053	1.25×10^8	**2.92**	25.59	143455	2.11×10^{15}	2.95	**1.81**	3103	3.72×10^9
	1.3	331574	**1.73**	16.25	92334	1.02×10^9	**1.70**	18.09	154449	1.41×10^{16}	**1.60**	1.74	5861	1.36×10^{11}
	1.4	310410	**1.21**	12.45	105507	4.54×10^9	**1.30**	14.03	179186	1.02×10^{17}	**1.28**	1.55	5710	1.54×10^{12}
	1.5	291785	**0.73**	8.88	98231	1.25×10^{10}	**0.74**	9.38	149403	3.06×10^{17}	**0.70**	1.21	5855	6.74×10^{12}
G_3	1.1	377742	**83.76**	1008.11	0	0	**77.19**	811.03	0	0	78.68	**66.96**	0	0
	1.2	348159	**32.87**	852.47	6641	2.32×10^5	**27.12**	657.89	17252	1.34×10^{13}	**27.27**	89.75	0	0
	1.3	322874	**23.33**	626.94	261978	3.12×10^{10}	**20.87**	452.10	768876	1.53×10^{19}	**36.20**	36.30	0	0
	1.4	301013	**12.08**	386.91	328581	4.92×10^{11}	**10.88**	266.19	917102	3.23×10^{20}	**9.70**	22.14	0	0
	1.5	281924	**10.81**	315.40	405816	3.02×10^{12}	**9.29**	205.90	1062331	9.94×10^{20}	**7.64**	19.44	606	2.88×10^{10}
G_4	1.1	402370	155.05	OOM	190520	1.54×10^{10}	**64.12**	1032.53	374111	5.43×10^{18}	51.95	**0.65**	0	0
	1.2	370499	**86.91**	628.93	739356	1.98×10^{14}	**24.09**	317.44	1374522	1.41×10^{23}	20.82	**0.96**	0	0
	1.3	343307	125.06	408.97	1148330	1.98×10^{16}	**14.83**	190.25	2005760	7.27×10^{24}	11.69	**1.48**	0	0
	1.4	319833	**108.25**	281.81	1465722	6.32×10^{17}	**12.18**	134.15	2495000	1.87×10^{26}	8.31	**3.09**	5645	2.19×10^{11}
	1.5	299363	**29.13**	190.59	1761682	1.65×10^{19}	**9.60**	85.84	2434632	4.02×10^{27}	5.55	**3.46**	15587	9.56×10^{14}
G_5	1.1	388844	> 1 h	OOM	-	-	> 1 h	OOM	-	-	> 1 h	**< 0.01**	0	0
	1.2	362027	> 1 h	OOM	-	-	> 1 h	OOM	-	-	> 1 h	**< 0.01**	0	0
	1.3	338670	OOM	OOM	-	-	> 1 h	OOM	-	-	> 1 h	**< 0.01**	0	0
	1.4	318145	OOM	OOM	-	-	OOM	OOM	-	-	OOM	**< 0.01**	0	0
	1.5	299965	OOM	**1960.28**	393178	9.20×10^{13}	OOM	OOM	-	-	OOM	**< 0.01**	0	0

■ **Table 3** Detailed experimental results of the proposed algorithm for G_3 (Chiba) and G_4 (Aichi) when the input graph partitions are induced partitions.

	r	$Z_{\mathcal{S}}$			$T_{\mathcal{S}\pm}$		$Z_{\mathcal{S}\uparrow}$			$Z_{\mathcal{A}} \setminus Z_{\mathcal{S}\uparrow}$
		time	node	card	time	node	time	node	card	time
G_3	1.1	1.90	54745	4.24×10^8	0.93	99057	75.88	2117874	2.17532×10^{40}	5.05
	1.2	1.01	39845	1.67×10^8	0.69	75581	27.94	977840	2.17528×10^{40}	3.23
	1.3	0.58	31030	6.62×10^7	0.51	60034	18.83	814538	2.17498×10^{40}	3.41
	1.4	0.34	24066	3.30×10^7	0.38	48818	8.49	490753	2.17490×10^{40}	2.87
	1.5	0.25	19877	1.42×10^7	0.34	40340	7.23	410152	2.17486×10^{40}	2.99
G_4	1.1	0.02	2376	2.09×10^4	0.32	11109	80.03	3074734	1.19200×10^{52}	74.68
	1.2	0.01	1686	1.03×10^4	0.20	8511	22.24	1205320	1.19174×10^{52}	64.46
	1.3	0.01	1235	6.11×10^3	0.17	6935	11.51	692798	1.19170×10^{52}	113.37
	1.4	< 0.01	961	3.67×10^3	0.14	5808	8.30	529214	1.19164×10^{52}	99.81
	1.5	< 0.01	756	2.67×10^3	0.13	4930	5.30	348832	1.19153×10^{52}	23.70

uses the input ZDD only in the set difference operation, which is executed in the last of the algorithm (by the existing apply-like method). As we show later, the bottleneck of Alg. N is the construction of $Z_{\mathcal{S}\uparrow}$. Therefore, in many cases, the sizes of input ZDDs do not change the execution time of Alg. N.

Third, we examine the results when the input graph partitions are rooted forests. There are 13 cases such that Alg. K was faster than Alg. N. In the cases, the sizes of input ZDDs and output ZDDs are small, that is, thousands, or even zero. These results show that Alg. K tends to be faster when the sizes of input ZDDs and output ZDDs are small.

In order to assess the efficiency of our algorithm in each step, we show detailed experimental results for G_3 and G_4 when the input graph partitions are induced partitions in Tab. 3. In the table, we show the time to construct decision diagrams, the size of decision diagrams, and the cardinality of the family represented by ZDDs. The cardinality of $S^{\pm}$ is omitted because it is equal to that of $\mathcal{S}$. The size and cardinality for $Z_{\mathcal{A}} \setminus Z_{\mathcal{S}\uparrow}$ are also omitted because they are the same as $|Z_{\mathcal{B}}|$ and $|\mathcal{B}|$, which are shown in Tab. 2. For both G_3 and G_4, the time to construct $Z_{\mathcal{S}}$ and $T_{\mathcal{S}\pm}$ are within one or two seconds. The most time-consuming parts are the construction of $Z_{\mathcal{S}\uparrow}$ in G_3 and $Z_{\mathcal{S}\uparrow}$ or $Z_{\mathcal{A}} \setminus Z_{\mathcal{S}\uparrow}$ in G_4. The set difference operation in G_4 took a lot of time because the sizes of $Z_{\mathcal{A}}$ and $Z_{\mathcal{S}\uparrow}$ are large, that is, more than a hundred. The reason why the construction of $Z_{\mathcal{S}\uparrow}$ takes a lot of time is the increase in the sizes of decision diagrams. While the size of $T_{\mathcal{S}\pm}$ is only 2–7 times larger than that of $Z_{\mathcal{S}}$, that of $Z_{\mathcal{S}\uparrow}$ is about 10–276 times larger than that of $T_{\mathcal{S}\pm}$. This also made the execution of the algorithm in G_5 impossible.

5 Conclusion

In this paper, we have proposed an algorithm to construct a ZDD representing all the graph partitions such that all the weights of its connected components are at least a given value. As shown in the experimental results, the proposed algorithm has succeeded in constructing a ZDD representing a set of more than 10^{12} graph partitions in ten seconds, which is 30 times faster than the existing state-of-the-art algorithm. Future work is devising a more memory efficient algorithm that enables us to deal with larger graphs, that is, graphs with hundreds of vertices. It is also important to seek for efficient algorithms to deal with other constraints on weights such that the ratio of the maximum and the minimum of weights is at most a specified value.

────── **References** ──────

1 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

2 Gary Hardy, Corinne Lucet, and Nikolaos Limnios. K-terminal network reliability measures with binary decision diagrams. *IEEE Transactions on Reliability*, 56(3):506–515, 2007.

3 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*, E82-A:714–721, 1999.

4 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, 2014.

5 Hiroaki Iwashita and Shin-ichi Minato. Efficient top-down ZDD construction techniques using recursive specifications. *TCS Technical Reports*, TCS-TR-A-13-69, 2013.

6 Jun Kawahara, Takashi Horiyama, Keisuke Hotta, and Shin-ichi Minato. Generating all patterns of graph partitions within a disparity bound. In *Proc. of the 11th International Conference and Workshops on Algorithms and Computation (WALCOM)*, pages 119–131, 2017.

7 Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 100(9):1773–1784, 2017.

8 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.

9 Jun Kawahara, Toshiki Saitoh, Hirofumi Suzuki, and Ryo Yoshinaka. Enumerating all subgraphs without forbidden induced subgraphs via multivalued decision diagrams. *CoRR*, 2018. `arXiv:1804.03822`.

10 Donald E. Knuth. *The art of computer programming, Vol. 4A, Combinatorial algorithms, Part 1*. Addison-Wesley, 2011.

11 Takanori Maehara, Hirofumi Suzuki, and Masakazu Ishihata. Exact computation of influence spread by binary decision diagrams. In *Proc. of the 26th International World Wide Conference (WWW)*, pages 947–956, 2017.

12 Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th ACM/IEEE design automation conference*, pages 272–277, 1993.

13 Yu Nakahata, Jun Kawahara, Takashi Horiyama, and Shoji Kasahara. Enumerating all spanning shortest path forests with distance and capacity constraints. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences (to appear).

14 Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the Tutte polynomial of a graph of moderate size. In *Proc. of the 6th International Symposium on Algorithms and Computation (ISAAC)*, pages 224–233, 1995.

15 Koichi Yasuoka. A new method to represent sets of products: ternary decision diagrams. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 78(12):1722–1728, 1995.

16 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.