Enumerating All Subgraphs Under Given Constraints Using Zero-Suppressed Sentential Decision Diagrams

Yu Nakahata
Graduate School of Informatics, Kyoto University, Japan
nakahata.yu.27e@st.kyoto-u.ac.jp

Masaaki Nishino
NTT Communication Science Laboratories, NTT Corporation, Kyoto, Japan
masaaki.nishino.uh@hco.ntt.co.jp

Jun Kawahara
Graduate School of Informatics, Kyoto University, Japan
jkawahara@i.kyoto-u.ac.jp

Shin-ichi Minato
Graduate School of Informatics, Kyoto University, Japan
minato@i.kyoto-u.ac.jp

Abstract

Subgraph enumeration is a fundamental task in computer science. Since the number of subgraphs can be large, some enumeration algorithms exploit compressed representations for efficiency. One such representation is the Zero-suppressed Binary Decision Diagram (ZDD). ZDDs can represent the set of subgraphs compactly and support several poly-time queries, such as counting and random sampling. Researchers have proposed efficient algorithms to construct ZDDs representing the set of subgraphs under several constraints, which yield fruitful results in many applications. Recently, Zero-suppressed Sentential Decision Diagrams (ZSDDs) have been proposed as variants of ZDDs. ZSDDs can be smaller than ZDDs when representing the same set of subgraphs. However, efficient algorithms to construct ZSDDs are known only for specific types of subgraphs: matchings and paths.

We propose a novel framework to construct ZSDDs representing sets of subgraphs under given constraints. Using our framework, we can construct ZSDDs representing several sets of subgraphs such as matchings, paths, cycles, and spanning trees. We show the bound of sizes of constructed ZSDDs by the branch-width of the input graph, which is smaller than that of ZDDs by the path-width. Experiments show that our methods can construct ZSDDs faster than ZDDs and that the constructed ZSDDs are smaller than ZDDs when representing the same set of subgraphs.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Subgraph, Enumeration, Decision Diagram, Zero-suppressed Sentential Decision Diagram (ZSDD), Top-down construction algorithm

Digital Object Identifier 10.4230/LIPIcs.SEA.2020.9

Funding This work was supported by JSPS KAKENHI Grant Number JP15H05711, JP18H04091, JP18K04610, and JP19J21000.

1 Introduction

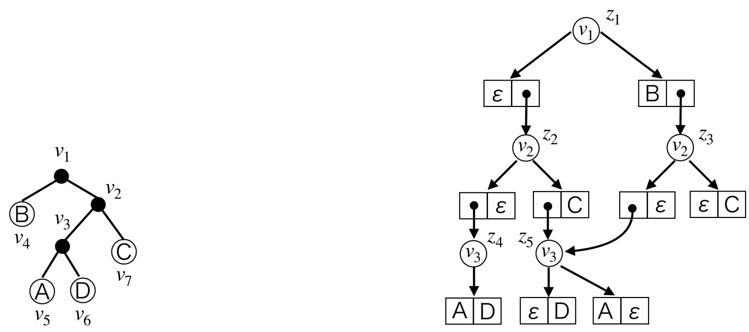
Enumerating subgraphs of a given graph under some constraint is a fundamental task in computer science. There are enumeration algorithms for several types of subgraphs such as cliques [5], paths [1], and spanning trees [21]. These algorithms list all subgraphs one by one in a small amount of time per subgraph. However, such algorithms take at least linear time and space to the number of subgraphs. Since the number of subgraphs can be exponentially larger than the size of the input graph, it is trouble when applied to practical problems.

Instead of explicitly listing subgraphs, some algorithms exploit compressed representations of sets of subgraphs. One such representation is the Zero-suppressed Binary Decision Diagram (ZDD) [15]. ZDDs are compact representations of set families. By regarding a subgraph as its edge set, we can express a set of subgraphs by a ZDD. ZDDs can not only represent set families compactly but also support several poly-time queries such as counting and random sampling [15]. In addition, given two ZDDs, we can efficiently construct a ZDD representing the union or intersection of the set families represented by the input ZDDs in polynomial time to the sizes of the input ZDDs. Such operations are called *Apply operations* [4]. Due to these merits, ZDDs appear in several graph-related applications such as network optimization [10], network reliability evaluation [7, 8], and balanced graph partitions [12, 16, 17].

A key to use ZDDs effectively for subgraph enumeration is a fast algorithm to construct a ZDD representing the set of subgraphs. It is time-consuming to construct a ZDD by first explicitly listing subgraphs and then combining ZDDs using Apply operations. In contrast, some algorithms can construct ZDDs *without explicitly listing subgraphs*. Such algorithms are called *top-down construction algorithms*, while algorithms using Apply operations are called *bottom-up*. Researchers have proposed top-down construction algorithms for ZDDs representing several sets of subgraphs [20, 7, 14]. Kawahara et al. [13] generalized the algorithms to obtain a general framework of top-down construction algorithms for ZDDs. Using the framework, we can construct ZDDs representing the sets of subgraphs under several constraints, such as the number of edges, degrees of vertices, and connectivity of vertices. By combining these fundamental constraints, we can specify several types of subgraphs, such as matchings, paths, cycles, and spanning trees.

Recently, Zero-suppressed Sentential Decision Diagrams (ZSDDs) [18] have been proposed as different representations of set families. Since ZSDDs are generalizations of ZDDs, ZSDDs are at least as compact as ZDDs. In theory, there exist set families that have polynomial ZSDD sizes but exponential ZDD sizes [3]. In addition, ZSDDs inherit some poly-time queries of ZDDs: counting, random sampling, and Apply operations. Thus, a natural question is: Can we design top-down construction algorithms for ZSDDs representing sets of subgraphs? The question is partially answered in an affirmative way by Nishino et al. [19]. They proposed top-down construction algorithms for ZSDDs representing sets of specific types of subgraphs: matchings and paths. The sizes of constructed ZSDDs by their algorithms are bounded by the *branch-width* of the input graph [19], while those of ZDDs are bounded by the *path-width* [11]. Since the branch-width of a graph never exceeds the path-width [2], ZSDDs have tighter upper bounds than ZDDs. The efficiency of their algorithms was confirmed in experiments. Despite such striking results, their algorithms are specific to matchings and paths.

In this paper, we propose a novel framework of top-down construction algorithms for ZSDDs. To design a top-down construction algorithm using our framework, one only has to prove a recursive formula for the desired set of subgraphs. Using the recursive formula, we can theoretically show the correctness and the complexity of the algorithm, which was difficult with the existing method. We apply our framework to the three fundamental constraints used in ZDDs: the number of edges, degrees of vertices, and connectivity of vertices. We show that the sizes of constructed ZSDDs are bounded by the branch-width of the input graph, not only for matchings and paths. Experiments show that proposed methods can construct ZSDDs faster than ZDDs and that the constructed ZSDDs are smaller than ZDDs representing the same sets of subgraphs.



(a) A vtree. **(b)** A ZSDD.

Figure 1 A vtree and a ZSDD that respects the vtree.

2 Preliminaries

2.1 Graphs

Let $G = (V, E)$ be an undirected graph where V is the vertex set and E is the edge set. $|V|$ and $|E|$ denote the number of vertices and edges, respectively. For edge subset $S \subseteq E$, the *induced subgraph* $G[S]$ is the subgraph $(V[S], S)$, where $V[S] \subseteq V$ is the set of vertices to which an edge in S is incident. In the following, we identify S with $G[S]$. For $S \subseteq E$ and $u \in V$, the *degree* $\deg(S, u)$ of u in S is the number of edges incident to u in S.

2.2 $(\mathbf{X}, \mathbf{Y})$-partition and vtree

Let f and g be set families. We define three operations between set families. We define *union* $\cup$, *intersection* $\cap$, and *join* $\sqcup$ as $f \cup g = \{a \mid a \in f \text{ or } a \in g\}$, $f \cap g = \{a \mid a \in f \text{ and } a \in g\}$, and $f \sqcup g = \{a \cup b \mid a \in f \text{ and } b \in g\}$, respectively.

▶ **Definition 1.** *Let f be a set family, and $\mathbf{X}, \mathbf{Y}$ be a partition of the universe of f. Set family f can be written as*

$$f = \bigcup_{i=1}^{h} [p_i \sqcup s_i], \tag{1}$$

where p_i and s_i are the set families whose universes are $\mathbf{X}$ and $\mathbf{Y}$, respectively. The equation is an $(\mathbf{X}, \mathbf{Y})$-decomposition. *We call $p_1, \dots, p_h$* primes *and $s_1, \dots, s_h$* subs. *If the primes are* exclusive *($p_i \cap p_j = \emptyset$ for all $i \neq j$), the decomposition is an* $(\mathbf{X}, \mathbf{Y})$-partition. [1]

▶ **Example 2.** Let f_1 be the family of subsets of $U_1 = \{A, B, C, D\}$ that contain exactly two elements. It follows that $f_1 = \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}\}$. For $\mathbf{X}_1 = \{B\}$ and $\mathbf{Y}_1 = \{A, C, D\}$, an $(\mathbf{X}_1, \mathbf{Y}_1)$-partition of f_1 is

$$f_1 = [\underbrace{\{\emptyset\}}_{\text{prime}} \sqcup \underbrace{f_2^1}_{\text{sub}}] \cup [\underbrace{\{\{B\}\}}_{\text{prime}} \sqcup \underbrace{f_2^2}_{\text{sub}}], \tag{2}$$

where $f_2^1 = \{\{A, C\}, \{A, D\}, \{C, D\}\}$ and $f_2^2 = \{\{A\}, \{C\}, \{D\}\}$.

[1] In [18], an $(\mathbf{X}, \mathbf{Y})$-decomposition is called an $(\mathbf{X}, \mathbf{Y})$-partition if the primes are exclusive and *consistent* ($p_i \neq \emptyset$ for all i). For simplicity, we do not require consistency for $(\mathbf{X}, \mathbf{Y})$-partitions. If we construct a ZSDD without consistency, we can make their primes consistent in linear time to the ZSDD size [19].

The universe of f_2^1 and f_2^2 is $U_2 = \{A, C, D\}$. For $\mathbf{X}_2 = \{A, D\}$ and $\mathbf{Y}_2 = \{C\}$, an $(\mathbf{X}_2, \mathbf{Y}_2)$-partition of f_2^1 is

$$f_2^1 = [\underbrace{\{\{A, D\}\}}_{\text{prime}} \sqcup \underbrace{\{\emptyset\}}_{\text{sub}}] \cup [\underbrace{\{\{A\}, \{D\}\}}_{\text{prime}} \sqcup \underbrace{\{\{C\}\}}_{\text{sub}}]. \tag{3}$$

A ZSDD represents a set family by recursively applying $(\mathbf{X}, \mathbf{Y})$-partitions to decompose the family into sub-families, where the order of partitions is determined by a *vtree*. A vtree is a rooted, ordered, and full binary tree whose leaves correspond to elements of the universe. Figure 1a shows an example. Symbols appearing in leaves represent corresponding elements, and symbols beside nodes represent their names. Each internal node represents a partition of the universe into two subsets: elements appearing in the left and right subtrees. We denote the left and right children of node v by v^l and v^r, respectively. In the figure, root node v_1 represents the $(\mathbf{X}_1, \mathbf{Y}_1)$-partition of the universe $U_1 = \{A, B, C, D\}$ where $\mathbf{X}_1 = \{B\}$ and $\mathbf{Y}_1 = \{A, C, D\}$. Similarly, node v_2 represents the $(\mathbf{X}_2, \mathbf{Y}_2)$-partition of the universe $U_2 = \{A, C, D\}$ where $\mathbf{X}_2 = \{A, D\}$ and $\mathbf{Y}_2 = \{C\}$. To avoid confusion, we call vtree nodes *vnodes*, ZSDD nodes *znodes*, and graph nodes *vertices*. We represent them as v_i, z_i, and u_i.

2.3 Zero-suppressed Sentential Decision Diagrams

A ZSDD is recursively defined as follows. ZSDD α *respects* vnode v if the order of $(\mathbf{X}, \mathbf{Y})$-partitions in α follows the vtree whose root is v. $\langle \alpha \rangle$ denotes the set family that α represents.

▶ **Definition 3.** *α is a ZSDD that respects vnode v if and only if:*

- *$\alpha = \varepsilon$ or $\alpha = \perp$. (Semantics: $\langle \varepsilon \rangle = \{\emptyset\}$ and $\langle \perp \rangle = \emptyset$.)*
- *$\alpha = X$ or $\alpha = \pm X$ and v is a leaf with element X. (Semantics: $\langle X \rangle = \{\{X\}\}$ and $\langle \pm X \rangle = \{\{X\}, \emptyset\}$.)*
- *$\alpha = \{(p_1, s_1), \ldots, (p_h, s_h)\}$, v is internal, $p_1, \ldots, p_h$ are ZSDDs that respect a vnode in the subtree whose root is v^l, $s_1, \ldots, s_h$ are ZSDDs that respect a vnode in the subtree whose root is v^r, and $\langle p_1 \rangle, \ldots, \langle p_h \rangle$ are exclusive. (Semantics: $\langle \alpha \rangle = \bigcup_{i=1}^{h} [\langle p_i \rangle \sqcup \langle s_i \rangle]$.)*

If a ZSDD is either $\varepsilon, \perp, X$, or $\pm X$, it is a *terminal*. Otherwise, it is a *decomposition*. Figure 1b shows an example ZSDD that represents set family f_1 in Example 2 and respects the vtree in Figure 1a. A circle node and its child rectangle nodes represent an $(\mathbf{X}, \mathbf{Y})$-partition. The symbol in a circle node indicates the vnode that the decomposition respects. A pair of rectangle nodes represent a prime-sub pair in an $(\mathbf{X}, \mathbf{Y})$-partition where the left and right are prime p and sub s, respectively. Every p and s is either a terminal ZSDD or a pointer to a decomposition ZSDD. Circle nodes are *decomposition znodes*, and rectangle nodes are *element znodes*. For example, znodes z_1 and z_2 represent the $(\mathbf{X}, \mathbf{Y})$-partitions in Equations (2) and (3), respectively. The *size* of a ZSDD is the sum of the sizes of $(\mathbf{X}, \mathbf{Y})$-partitions in the ZSDD. The size of the ZSDD in Figure 1b is 9. [2]

3 A novel framework of top-down ZSDD construction

We present a novel framework of top-down ZSDD construction. Our framework is partially identical to that of Nishino et al.'s [19], but we modify it so that we can design algorithms easily for several constraints. Algorithm 1 shows the framework. The algorithm takes graph G and the root vnode as its inputs and returns a ZSDD representing a set of subgraphs

[2] The size of a ZDD is defined as the number of nodes. [15] This is because, every node of a ZDD has exactly two children. In contrast, nodes of a ZSDD may have different number of children, and thus the size of a ZSDD is defined as the number of arcs.

Algorithm 1 A top-down construction algorithm.

Input : A graph $G = (V, E)$ and the root vtree node v
Output : A ZSDD representing a set of subgraphs of G

$Z[v] \leftarrow \mathsf{rootState}()$
$\mathsf{construct}(v, Z)$
$Z \leftarrow \mathsf{reduce}(Z)$
return Z

Algorithm 2 $\mathsf{construct}(v, Z)$.

1 **for** $z \in Z[v]$ **do**
2   `elems` $\leftarrow \emptyset$
3   **for** $(m^l, m^r) \in \mathsf{decomp}(v, z)$ **do**
4   **for** $\circ \in \{l, r\}$ **do**
5   **if** v° *is a leaf vnode* **then** $z^\circ \leftarrow \mathsf{terminal}(v^\circ, m^\circ)$
6   **else** $z^\circ \leftarrow \mathsf{unique}(v^\circ, m^\circ, Z)$
7   `elems` $\leftarrow$ `elems` $\cup \left\{(z^l, z^r)\right\}$
8   Set `elems` as the child znodes of z
9   **for** $\circ \in \{l, r\}$ **do**
10   **if** v° *is an internal vnode* **then** $\mathsf{construct}(v^\circ, Z)$

of G. $Z[v]$ stores a set of decomposition znodes that respect vnode v. Since a ZSDD is represented as a set of decomposition znodes, the set of $Z[v]$'s for all internal vnodes v can be seen as a ZSDD. The algorithm first calls $\mathsf{rootState}()$, which returns the root znode. The procedure depends on the types of subgraphs. The algorithm next calls $\mathsf{construct}(v, Z)$, which recursively construct child znodes of znodes respecting v. If we naively construct znodes, the number of child znodes grows exponentially. We thus merge *equivalent* znodes during the construction of a ZSDD. Here, two znodes are equivalent if they respect the same vnode and represent the same family of sets. To detect equivalent znodes efficiently, we attach a *label* to each znode. The labels must be defined depending on the types of subgraphs so that two znodes are equivalent if they respect the same vnode and have the same label. We explain how to design labels in Section 4. The constructed ZSDD may have redundant znodes. Function $\mathsf{reduce}(Z)$ deletes such znodes.

Algorithm 2 shows function $\mathsf{construct}(v, Z)$. The function is called only for internal vnodes. In [19], the procedure of $\mathsf{construct}(v, Z)$ was designed depending on whether v^l is a leaf or not. Instead, we treat all internal vnodes in the same way, which makes it easy to design algorithms for several constraints. For each znode z in $Z[v]$, the function calculates the prime-sub pairs corresponding to z. We first initialize the set of prime-sub pairs `elems` to the empty set (Line 2). Function $\mathsf{decomp}(v, z)$ receives vnode v and znode z that respects v, and returns the set of pairs of labels corresponding to the prime-sub pairs (Line 3). For each $\circ \in \{l, r\}$, if v° is a leaf vnode, we set znode z° to a terminal (Line 5). Function $\mathsf{terminal}(v, m)$ receives leaf vnode v and label m, and returns an appropriate terminal depending on the types of subgraphs. If v° is an internal vnode, we call $\mathsf{unique}(v, m, Z)$ (Line 6). The function receives vnode v and label m, and checks whether $Z[v]$ contains a znode with label m. If such a znode exists, the function returns its address. Otherwise, the function creates a new znode that respects v and has label m, stores it into $Z[v]$, and returns its address. We add the prime-sub pair (z^l, z^r) into `elems` (Line 7). After generating all the prime-sub pairs, we set `elems` as the child znodes of z (Line 8). Finally, for each $\circ \in \{l, r\}$ such that v° is an internal vnode, we call $\mathsf{construct}(v^\circ, Z)$ to recursively construct sub-ZSDDs (Lines 9–10).

The functions $\mathsf{reduce}(Z)$ and $\mathsf{unique}(v, m, Z)$ can be designed regardless of the types of subgraphs [19]. In contrast, the definition of labels and the procedures of $\mathsf{rootState}()$, $\mathsf{terminal}(v, m)$, and $\mathsf{decomp}(v, z)$ heavily depend on the types of subgraphs. To easily design them for several constraints, we relate a recursive formula for the desired set of subgraphs to top-down ZSDD construction. Intuitively, in our framework, internal vnodes correspond to recursion steps, while leaf vnodes correspond to base cases. Therefore, we only have to prove a recursive formula for the desired set of subgraphs. The recursive formula directly leads to the definition of labels and the procedures of subroutines. We can also show the correctness of the algorithm and the bound of the constructed ZSDD size from the recursive formula.

4 Subroutines for several constraints

We apply our framework to three fundamental constraints: the number of edges, degrees of vertices, and connectivity of vertices. By combining these constraints, we can specify several types of subgraphs. For each constraint, we show a recursive formula for the set of subgraphs satisfying the constraint. Using the recursive formula, we derive subroutines and bound the sizes of constructed ZSDDs. The proofs are omitted due to the space limitation.

4.1 Cardinality

Given graph $G = (V, E)$, vtree T whose leaves are labeled by the elements of E, and non-negative integer k^*, we construct a ZSDD that represents the family of sets with exactly k^* elements. We can also construct a ZSDD that represents the family of sets with at most or at least k^* elements (details are omitted). In the following, we focus on the "exactly k^*" constraint. For vnode v, let $E(v) \subseteq E$ be the set of graph edges that correspond to the leaf vnodes of the sub-vtree whose root is v. For vnode v and non-negative integer k, let $f(v, k)$ be the family of subsets of $E(v)$ with k elements, that is, $f(v, k) = \{S \mid S \subseteq E(v), |S| = k\}$. The desired family is $f(v^{\mathrm{root}}, k^*)$, where v^{root} is the root vnode of T. For leaf vnode v, $\ell(v)$ denotes the element corresponding to v. We show a recursive formula for $f(v, k)$.

▶ **Lemma 4.** *Let v be a vnode, and k be a non-negative integer. If v is a leaf vnode, then the following hold:*

$$f(v, k) = \begin{cases} \{\emptyset\} & (k = 0) \\ \{\{\ell(v)\}\} & (k = 1) \\ \emptyset & (\text{otherwise}) \end{cases} \tag{4}$$

If v is internal, the following is an $(E(v^l), E(v^r))$-partition:

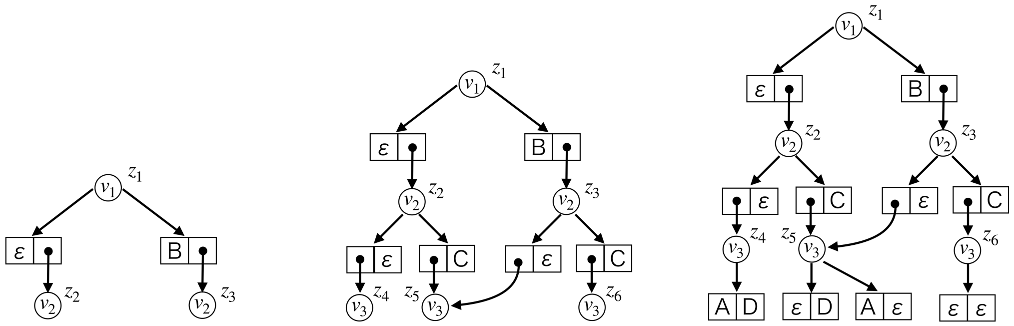
$$f(v, k) = \bigcup_{i=0}^{k} \left[f(v^l, i) \sqcup f(v^r, k - i) \right]. \tag{5}$$

Using the recursive formula, we can design the subroutines of the framework. We use non-negative integers as znode labels. For znode z that respects vnode v, the label of z indicates the number of elements that should be adopted from $E(v)$. Function $\mathsf{rootState}()$ returns the root znode with label k^*, since the desired family is $f(v^{\mathrm{root}}, k^*)$. Algorithm 3 shows the subroutines $\mathsf{terminal}(v, k)$ and $\mathsf{decomp}(v, z)$. $\mathsf{terminal}(v, k)$ is obtained from Equation (4). If $k = 0$, it returns ε since $\langle \varepsilon \rangle = \{\emptyset\}$ (Line 1). If $k = 1$, it returns $\ell(v)$ since $\langle \ell(v) \rangle = \{\{\ell(v)\}\}$ (Line 2). Otherwise, it returns $\perp$ since $\langle \perp \rangle = \emptyset$ (Line 3). Similarly, $\mathsf{decomp}(v, z)$ is obtained from Equation (5). The function initializes `elems` to the empty set (Line 4). Let k be the

Algorithm 3 Subroutines for the cardinality constraint.

```
Function :terminal(v, k)
1 if k = 0 then return ε
2 else if k = 1 then return ℓ(v)
3 else return ⊥

Function :decomp(v, z)
4 elems ← ∅
5 Let k be the label of z
6 for i ∈ [0, k] do
7     elems ← elems ∪ {(i, k − i)}
8 return elems
```



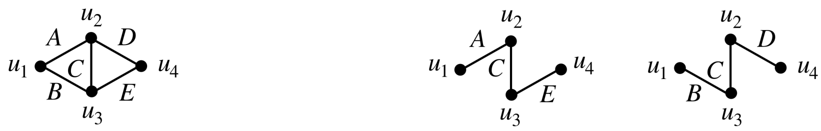
(a) After construct(v_1, Z). **(b)** After construct(v_2, Z). **(c)** After construct(v_3, Z).

Figure 2 Intermediate ZSDDs for the cardinality constraint.

label of z (Line 5). If the prime has label $0 \leq i \leq k$, then the sub has label $k - i$. Thus, we add the pair $(i, k - i)$ to `elems` (Lines 6–7). Finally, we return `elems` (Line 8). The correctness of the algorithm directly follows from the correctness of Lemma 4.

▶ **Example 5.** Let us construct a ZSDD that represents the family of subsets of $\{A, B, C, D\}$ with exactly two elements. We use the vtree in Figure 1a. First, rootState() creates root znode z_1 with label 2 and stores it into $Z[v_1]$. The function then calls construct(v_1, Z). $Z[v_1]$ contains only one znode z_1. Since z_1 has label 2, decomp(v_1, z_1) returns $\{(0, 2), (1, 1), (2, 0)\}$. The function first processes label pair $(0, 2)$. Since $v_1^l = v_4$ is a leaf vnode, the function calls terminal$(v_4, 0)$, which returns ε. Since $v_1^r = v_2$ is not a leaf vnode, the function calls unique$(v_2, 2, Z)$. It creates new decomposition znode z_2 that respects v_4 and has label 2, stores it into $Z[v_4]$, and returns its address. Similarly, for label pair $(1, 1)$, the corresponding prime-sub pair is calculated as (B, z_3), where z_3 is a new decomposition znode that respects v_2 and has label 1. As for label pair $(2, 0)$, since the universe of the prime contains only one element, we discard this pair. As a result, the function set the prime-sub pairs (ε, z_2) and (B, z_3) as child znodes of z_1. Figure 2a shows the current intermediate ZSDD. Since $v_1^l = v_4$ is a leaf vnode and $v_1^r = v_2$ is an internal vnode, the function calls only construct(v_2, Z).

We go on to construct(v_2, Z). $Z[v_2]$ contains two znodes z_2 and z_3. The function processes z_2 first. Since z_2 has label 2, decomp(v_2, z_2) returns $\{(2, 0), (1, 1), (0, 2)\}$. However, $(0, 2)$ is discarded because the universe of the sub only contains one element. As a result, the prime-sub pairs are calculated as $\{(z_4, \varepsilon), (z_5, C)\}$, where z_4 and z_5 are new decomposition znodes that respect v_3. The labels of z_4 and z_5 are 2 and 1, respectively. The function processes z_3 next. decomp(v_2, z_3) returns $\{(1, 0), (0, 1)\}$. Here, znode z_5 with label 1 already exists in $Z[v_3]$, and thus unique$(v_3, 1, Z)$ returns z_5. As a result, the set of prime-sub pairs is $\{(z_5, \varepsilon), (z_6, C)\}$, where z_6 is a new znode that respects v_3 and has label 0. Figure 2b shows the current intermediate ZSDD. Finally, construct(v_3, Z) is called and Figure 2c shows the resulting ZSDD. By calling reduce(Z), the ZSDD can be trimmed as Figure 1b.



(a) A graph. **(b)** Subgraphs.

Figure 3 A graph and its subgraphs satisfying a degree constraint.

Using Lemma 4, we can also bound the size of the constructed ZSDD.

▶ **Theorem 6.** *If α is the ZSDD obtained by Algorithm 3, the size of α is $\mathcal{O}(|E|k^2)$.*

4.2 Degree

We denote a given degree constraint by function $\delta^*\colon V \to \mathbb{N}$, where $\mathbb{N}$ is the set of non-negative integers. For subgraph $S \subseteq E$, we say that S *satisfies* δ^* if $\deg(S,u) = \delta^*(u)$ holds for all $u \in V$. For example, for the graph shown in Figure 3a and degree constraint δ^* such that $\delta^*(u_1) = \delta^*(u_4) = 1$ and $\delta^*(u_2) = \delta^*(u_3) = 2$, there are two subgraphs satisfying δ^* as shown in Figure 3b. Given G, T, and δ^*, we construct a ZSDD representing the set of all subgraphs satisfying δ^*. When a subgraph satisfies δ^*, for every vertex u, the degree of u in a subgraph must be "exactly" $\delta^*(u)$. Although we mainly discuss this "exact" constraint, we can easily modify the algorithm to deal with "at most" or "at least" constraints.

Similarly to Lemma 4, we show a recursive formula for the set of subgraphs satisfying the degree constraint. For vnode v, $V(v)$ denotes the set of vertices to which an edge in $E(v)$ is incident. Let us consider a degree constraint whose domain is limited to $V(v)$ as function $\delta\colon V(v) \to \mathbb{N}$. We define $f(v,\delta)$ as the family of subsets of $E(v)$ such that, for all $u \in V(v)$ and $S \in f(v,\delta)$, degree $\deg(S,u)$ equals $\delta(u)$. We show a recursive formula for $f(v,\delta)$.

▶ **Lemma 7.** *Let v be a vnode, and δ be a function from $V(v)$ to $\mathbb{N}$. If v is a leaf vnode, let u_1 and u_2 be the endpoints of graph edge $\ell(v)$. Then, the following hold:*

$$f(v,\delta) = \begin{cases} \{\emptyset\} & (\delta(u_1) = \delta(u_2) = 0) \\ \{\{\ell(v)\}\} & (\delta(u_1) = \delta(u_2) = 1) \\ \emptyset & (\text{otherwise}) \end{cases} \tag{6}$$

If v is internal, the following is an $(E(v^l), E(v^r))$-partition:

$$f(v,\delta) = \bigcup_{(\delta^l,\delta^r) \in P(v,\delta)} [f(v^l,\delta^l) \sqcup f(v^r,\delta^r)], \tag{7}$$

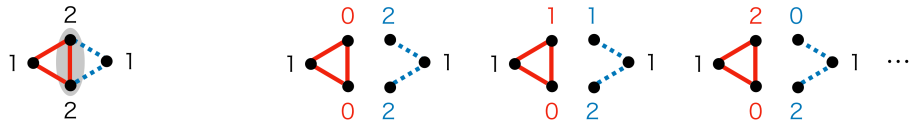
where $P(v,\delta)$ is the set of pairs of functions $\delta^l\colon V(v^l) \to \mathbb{N}$ and $\delta^r\colon V(v^r) \to \mathbb{N}$ such that

$$\forall u \in V(v^l) \cap V(v^r), \quad \delta^l(u) + \delta^r(u) = \delta(u), \tag{8}$$

$$\forall u \in V(v^l) \setminus V(v^r), \quad \delta^l(u) = \delta(u), \tag{9}$$

$$\forall u \in V(v^r) \setminus V(v^l), \quad \delta^r(u) = \delta(u). \tag{10}$$

For vnode v, the *frontier* of v is $F(v) = V(v^l) \cap V(v^r)$. Let us consider the graph shown in Figure 3a and the degree constraint δ^*, which we defined above. For vnode v, let $E(v^l) = \{A,B,C\}$ and $E(v^r) = \{D,E\}$. It follows that $F(v)$ is $\{u_2,u_3\}$. Figure 4a shows the current situation. The set of red (solid) and blue (dashed) edges are $E(v^l)$ and $E(v^r)$,



(a) A graph. **(b)** Prime-sub pairs.

Figure 4 A graph and corresponding prime-sub pairs.

respectively. The set of vertices in the shaded area is $F(v)$. We can interpret Equations (7) to (10) as follows. For vertex $u \in V(v^l) \setminus V(v^r)$, $\delta(u)$ edges in $E(v^l)$ must be incident to u, and thus $\delta^l(u) = \delta(u)$ (Equation (9)). A similar statement holds for vertices in $V(v^r) \setminus V(v^l)$ (Equation (10)). The remaining vertices are in $F(v)$. For vertex $u \in F(v)$, both edges in $E(v^l)$ and $E(v^r)$ are incident to u. Here, we guess how many edges in $E(v^l)$ are incident to u. This results in generating nine prime-sub pairs, as shown in Figure 4b. We can construct the ZSDD by recursively applying Lemma 7. Here we use δ as a label of a znode.

Let us analyze the sizes of ZSDDs constructed by our algorithm. The *width* of a vtree is $\max_{v \in \mathrm{in}(T)} V(v^l) \cap V(v^r)$, where $\mathrm{in}(T)$ is the set of internal vnodes.

▶ **Theorem 8.** *If α is the ZSDD representing $f(v^{\mathrm{root}}, \delta^*)$ obtained by our algorithm, the size of α is $\mathcal{O}(|E|d^{2W})$, where $d = \max_{u \in V} \delta^*(u) + 1$ and W is the width of the input vtree.*

There exists a vtree whose width equals the *branch-width* of the graph [19]. Given such a vtree, the ZSDD size is $\mathcal{O}(|E|d^{2\mathrm{bw}(G)})$, where $\mathrm{bw}(G)$ is the branch-width of G.

4.3 Spanning tree

We construct a ZSDD representing the set of all spanning trees of G. With a few modifications, we can also construct a ZSDD representing the set of all connected subgraphs. We introduce some notation. If vertices u, u' are connected in subgraph $S \subseteq E$, we write $u \overset{S}{\sim} u'$. Note that $\overset{S}{\sim}$ is an equivalence relation on V; an equivalence class (a set of vertices) is a *connected component* of S. Two vertex subsets $C, C' \subseteq V$ are *connected* if there exist $u \in C$ and $u' \in C'$ with $u \overset{S}{\sim} u'$; we write this as $C \overset{S}{\sim} C'$. We also write $u \overset{S}{\sim} C'$ if $C \overset{S}{\sim} C'$ for $C = \{u\}$.

For vnode v, let $\mathcal{C}$ be a partition of vertex set $F(v)$, that is, $\mathcal{C} = \{C_1, \ldots, C_g\}$ where $C_i \subseteq F(v)$ is a vertex set satisfying $C_i \cap C_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{g} C_i = F(v)$. Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a disjoint set family defined over vertex sets in $\mathcal{C}$, that is, $R_i \subseteq \mathcal{C}$ and $R_i \cap R_j = \emptyset$ for all $i \neq j$. Let $U(\mathcal{R}) = \{C \mid \exists i : C \in R_i\}$. Function $\mathsf{Same}(\mathcal{R}, C, C')$ returns `true` if there exists $R_i \in \mathcal{R}$ such that $C, C' \in R_i$, otherwise `false`. To represent the set of all spanning trees, we define $f(v, \mathcal{C}, \mathcal{R})$ as the set of subgraphs $S \subseteq E(v)$ satisfying the following:

- for every $C_1, C_2 \in U(\mathcal{R})$, $C_1 \overset{S}{\sim} C_2$ holds if and only if $\mathsf{Same}(\mathcal{R}, C_1, C_2) = \texttt{true}$,
- for every $C \in \mathcal{C} \setminus U(\mathcal{R})$, there exists a unique $C' \in U(\mathcal{R})$ such that $C \overset{S}{\sim} C'$. Similarly, for every $u \in V(v) \setminus F(v)$, there exists a unique $C' \in U(\mathcal{R})$ such that $u \overset{S}{\sim} C'$, and
- S does not contain a cycle.

Intuitively, $\mathcal{C}$ represents the sets of equivalent vertices. That is, vertices in the same vertex group $C \in \mathcal{C}$ are regarded to be connected. $\mathcal{R}$ represents the connectivity constraints over such equivalent sets of vertices. The first condition above requires that two vertex subsets C and C' must be connected in S if and only if they appear in the same $R \in \mathcal{R}$. The second condition requires that, every equivalent vertex subset appearing in $V(v)$ but does not appear in $\mathcal{R}$ must be connected to a vertex subset C' appearing in $\mathcal{R}$. The third

Algorithm 4 Subroutines for spanning trees.

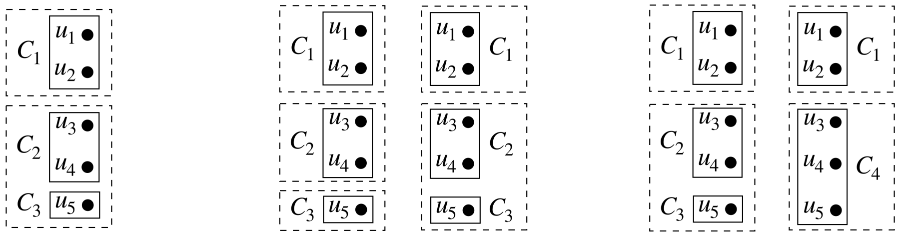
Function : $\mathsf{terminal}(v, (\mathcal{C}, \mathcal{R}))$
1 Let u_1 and u_2 be the endpoints of the graph edge $\ell(v)$
2 **if** $\mathsf{Same}(\mathcal{C}, u_1, u_2) = \texttt{true}$ **then**
3   Let $C \in \mathcal{C}$ be the set containing u_1 and u_2
4   **if** $C \in U(\mathcal{R})$ **then return** ε **else return** $\perp$
5 **else**
6   Let $C_1, C_2 \in \mathcal{C}$ be the sets containing u_1 and u_2, respectively
7   **if** *neither* C_1 *nor* C_2 *is in* $U(\mathcal{R})$ **then return** $\perp$
8   **else if** *exactly one of* C_1 *or* C_2 *is in* $U(\mathcal{R})$ **then return** $\ell(v)$
9   **else**
10   **if** $\mathsf{Same}(\mathcal{R}, C_1, C_2) = \texttt{true}$ **then return** $\ell(v)$ **else return** ε

Function : $\mathsf{decomp}(v, z)$
11 $\texttt{elems} \leftarrow \emptyset$
12 Let $(\mathcal{C}, \mathcal{R})$ be the label of z
13 $\mathcal{C}^l \leftarrow \left\{C \cap F(v^l) \;\middle|\; C \in \mathcal{C}, C \cap F(v^l) \neq \emptyset\right\} \cup \left\{\{u\} \;\middle|\; u \in F(v^l) \setminus F(v)\right\}$
14 **for** $\mathcal{R}^l \in \mathsf{enumPartition}(\mathcal{C}^l)$ **do**
15   **if** $\mathsf{isCompatible}(\mathcal{C}, \mathcal{R}, \mathcal{R}^l) = \texttt{true}$ **then**
16         $\mathcal{C}^r, \mathcal{R}^r \leftarrow \mathsf{calcSubState}(\mathcal{C}, \mathcal{R}, \mathcal{R}^l)$
17         $\texttt{elems} \leftarrow \texttt{elems} \cup \left\{((\mathcal{C}^l, \mathcal{R}^l), (\mathcal{C}^r, \mathcal{R}^r))\right\}$

condition is for acyclicity. The set of all spanning trees of G is $f(v^{\mathrm{root}}, \mathcal{C}^*, \mathcal{R}^*)$, where $\mathcal{C}^* = \{\{u\} \mid u \in F(v^{\mathrm{root}})\}$ and $\mathcal{R}^* = \{\{C\}\}$ for an arbitrary $C \in \mathcal{C}^*$ since initially there are no equivalent vertices and all vertices must be connected to form a spanning tree.

Unfortunately, it is quite complicated to show a recursive formula for $f(v, \mathcal{C}, \mathcal{R})$ and prove it theoretically. Thus, we show pseudo-code of subroutines and explain the behavior using an example. We use $(\mathcal{C}, \mathcal{R})$ as a znode label. $\mathsf{rootState}()$ returns the root znode label $(\mathcal{C}^*, \mathcal{R}^*)$. Algorithm 4 shows functions $\mathsf{terminal}(v, (\mathcal{C}, \mathcal{R}))$ and $\mathsf{decomp}(v, z)$. $\mathsf{terminal}(v, (\mathcal{C}, \mathcal{R}))$ returns an appropriate terminal with respect to the label of z. Let u_1 and u_2 be the endpoints of edge $\ell(v)$. We first consider the case that u_1 and u_2 are contained in the same vertex group $C \in \mathcal{C}$ (Lines 2–4). If $C \notin U(\mathcal{R})$, C must be connected to some $C' \in U(\mathcal{R})$. However, now we have $\mathcal{C} = \{C\}$, and thus there is no such C'. Therefore, we return $\perp$. If $C \in U(\mathcal{R})$, to avoid generating a cycle, we must not adopt edge $\ell(v)$. Thus we return ε. We next consider the case that u_1 and u_2 are contained in different sets $C_1, C_2 \in \mathcal{C}$ (Lines 5–10). If neither C_1 nor C_2 appear in constraints $\mathcal{R}$, they must be connected to some $C' \in U(\mathcal{R})$, but there are no such C'. Thus we return $\perp$ (Line 7). If either of C_1 or C_2 appears in $\mathcal{R}$, the unconstrained one must be connected with the other one, which has a constraint in $\mathcal{R}$. Thus we return $\ell(v)$ (Line 8). If both C_1 and C_2 appear in $\mathcal{R}$, we return the corresponding terminal depending on whether they appear in the same $R_i \in \mathcal{R}$ or not. If so, edge $\ell(v)$ must be adopted, and thus we return $\ell(v)$. Otherwise, the edge must not be adopted, and thus we return ε (Lines 9–10).

We go on to $\mathsf{decomp}(v, z)$. We first enumerate all possible set of constraints $\mathcal{R}^l$ of the prime. Since $\mathcal{R}^l$ is a partition of vertex groups $\mathcal{C}$, function $\mathsf{enumPartition}(\mathcal{C}^l)$ enumerates all partitions of $\mathcal{C}^l$. There may be partitions of $\mathcal{C}^l$ that are not *compatible* with $(\mathcal{C}, \mathcal{R})$; If $C_1 \in R_i$ and $C_2 \in R_j$ for $R_i, R_j \in \mathcal{R}$ where $i \neq j$, they must not appear in the same $R \in \mathcal{R}^l$. In addition, for every constraint $R \in \mathcal{R}^l$, a vertex in $F(v^l)$ must appear in some $C \in R$ in order to obtain a spanning tree. If both conditions are satisfied, $\mathcal{R}^l$ is compatible with $(\mathcal{C}, \mathcal{R})$. Function $\mathsf{isCompatible}(\mathcal{C}, \mathcal{R}, \mathcal{R}^l)$ returns $\texttt{true}$ if $\mathcal{R}^l$ is compatible with $(\mathcal{C}, \mathcal{R})$,



(a) A state. **(b)** Prime-sub pairs.

Figure 5 Label of the connectivity constraint and corresponding prime-sub pairs.

otherwise `false`. calcSubState$(\mathcal{C}, \mathcal{R}, \mathcal{R}^l)$ calculates $\mathcal{C}^r$ and $\mathcal{R}^r$ from its arguments. Intuitively, $\mathcal{C}^r$ and $\mathcal{R}^r$ are obtained by updating equivalent vertex groups in $\mathcal{C}$ by assuming constraints in $\mathcal{R}^l$ are satisfied. Let us give an example. Figure 5a shows a label and Figure 5b shows the corresponding prime-sub pairs. Five vertices $u_1, \ldots, u_5$ are on the frontier. We assume $F(v^l) = F(v^r) = F(v)$ in this example. In Figure 5a, the vertices are partitioned into three equivalency groups $\mathcal{C} = \{C_1, C_2, C_3\}$, where $C_1 = \{u_1, u_2\}$, $C_2 = \{u_3, u_4\}$, and $C_3 = \{u_5\}$. $\mathcal{C}$ is further partitioned into $\mathcal{R} = \{\{C_1\}, \{C_2, C_3\}\}$. $\mathcal{C}$ and $\mathcal{R}$ are depicted by solid and dashed rectangles, respectively. There are only two $\mathcal{R}^l$'s that are compatible with $(\mathcal{C}, \mathcal{R})$: $\mathcal{R}^l_1 = \{\{C_1\}, \{C_2\}, \{C_3\}\}$ and $\mathcal{R}^l_2 = \{\{C_1\}, \{C_2, C_3\}\}$. calcSubState$(\mathcal{C}, \mathcal{R}, \mathcal{R}^l_1)$ returns $(\mathcal{C}^r_1, \mathcal{R}^r_1)$, where $\mathcal{C}^r_1 = \{C_1, C_2, C_3\}$ and $\mathcal{R}^r_1 = \{\{C_1\}, \{C_2, C_3\}\}$. calcSubState$(\mathcal{C}, \mathcal{R}, \mathcal{R}^l_2)$ returns $(\mathcal{C}^r_2, \mathcal{R}^r_2)$, where $\mathcal{C}^r_2 = \{C_1, C_4\}$, $\mathcal{R}^r_2 = \{\{C_1\}, \{C_4\}\}$, and $C_4 = C_2 \cup C_3 = \{u_3, u_4, u_5\}$.

Finally, the following theorem states the bound of constructed ZSDD size.

▶ **Theorem 9.** *If α is a ZSDD representing the set of all spanning trees constructed by our top-down algorithm, the size of α is $\mathcal{O}(|E|W^{3W})$, where W is the width of the vtree.*

As discussed in Section 4.2, there exists a vtree whose width equals the branch-width of the graph. Given such a vtree, the size of a constructed ZSDD is $\mathcal{O}(|E|\mathrm{bw}(G)^{3\mathrm{bw}(G)})$.

5 Experiments

We conduct experiments to evaluate the performance of the proposed top-down construction algorithms for ZSDDs in the same way as an existing paper [19]. The vtrees for ZSDDs are obtained by a practical algorithm to find a branch decomposition with a small width [6]. To implement the top-down algorithm for ZDDs, we use the top-down algorithm for ZSDDs with a limitation that vtrees must be right-linear. Here, a vtree is *right-linear* if, for every internal vnode, its left child is a leaf. Since there is a one-to-one correspondence between ZDDs with ZSDDs using right-linear vtrees, by inputting right-linear vtrees, we can simulate ZDD construction. We use two element orders for ZDDs. The first one uses the order obtained by a breadth-first traversal of input graphs, as is used in graphillion [9], a library that implements a top-down construction algorithm for ZDDs. The other one uses the order induced from the vtrees used in the proposed method. Here we say an order is induced if a left-right traversal of a vtree gives the visiting order of variables [22]. We use the benchmark graphs of [19]: TSPLIB and RomeGraph. We constructed ZSDDs representing two types of subgraphs: 1) maximum degree at most two and 2) spanning trees. All code was written in C++ and compiled by g++-5.4.0 with -O3 option. All experiments were conducted on a machine with Intel Xeon W-2133 3.60 GHz CPU and 256 GB RAM.

Table 1 Results of constructing ZSDDs and ZDDs representing the set of all subgraphs whose maximum degrees are at most 2.

			Time (ms)			Size		
instance	$\lvert V\rvert$	$\lvert E\rvert$	TD	Z(b)	Z(v)	TD	Z(b)	Z(v)
att48	48	130	381	6801	2291	194786	1065745	507169
berlin52	52	145	1021	-	36354	807660	-	5229861
eil51	51	142	1012	247736	46524	774280	27277682	5974875
grafo10106	100	119	5	2617	16	2658	15461	7529
grafo10124	100	139	9237	-	40842	3060950	-	3283397
grafo10153	100	136	3784	-	4658	832943	-	561283
grafo10183	100	132	132	-	157837	80127	-	4088915
grafo10184	100	140	4981	-	119366	1006210	-	2002968
grafo10204	100	148	156529	-	303366	15712819	-	19847326
grafo10223	100	135	863	-	5956	330554	-	826121

Table 2 Results of constructing ZSDDs and ZDDs representing the set of all spanning trees.

			Time (ms)			Size		
instance	$\lvert V\rvert$	$\lvert E\rvert$	TD	Z(b)	Z(v)	TD	Z(b)	Z(v)
att48	48	130	3494	103871	3005	279613	5098205	387715
berlin52	52	145	11826	-	62706	937746	-	3194017
eil51	51	142	25828	-	94272	838254	-	7178190
ulysses22	22	56	39	3391	65	3036	520035	16762
grafo10106	100	119	28	221161	53	1756	836212	4057
grafo10183	100	132	2866	-	538878	224373	-	16414697
grafo10223	100	135	48563	-	128097	1009299	-	7313087
grafo10248	100	126	301	195249	672	16524	1617024	47605

Tables 1 and 2 show the results. In the tables, TD means the proposed method. Z(b) and Z(v) indicate top-down methods for ZDDs that employ breadth-first ordering and vtree traversing ordering, respectively. The empty fields indicate failure to complete within 600 seconds. We omit the instances for which all the methods finished within a second and at most one method finished within 600 seconds. In almost all cases, TD ran fastest and the sizes of ZSDDs are smaller than those of ZDDs. For example, for spanning trees (Table 2), the time of TD is up to 7898 times faster than Z(b), and 188 times faster than Z(v). The size of TD is up to 476 times smaller than Z(b) and 73 times smaller than Z(v). These results show the efficiency of our method. Using constructed ZDDs and ZSDDs, we can also enumerate subgraphs explicitly in polynomial time per subgraph [15, 18].

6 Concluding remarks

We have proposed a novel framework of algorithms for top-down ZSDD construction. We have shown the solid subroutines for three fundamental constraints: the number of edges, degree of vertices, and connectivity of vertices. We have shown the sizes of constructed ZSDDs can be bounded by the branch-width of the input graph. Experiments confirmed the efficiency of our method. Using Apply operations, we can combine several constraints. For example, we can extract connected subgraphs from ZSDD α by constructing ZSDD β representing the set of all connected subgraphs and computing $\alpha \cap \beta$. We believe that our framework can be used to solve various real-world problems.

References

1 Etienne Birmelé, Rui A. Ferreira, Roberto Grossi, Andrea Marino, Nadia Pisanti, Romeo Rizzi, and Gustavo Sacomoto. Optimal listing of cycles and st-paths in undirected graphs. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1884–1896. SIAM, 2013. doi:10.1137/1.9781611973105.134.

2 Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998. doi:10.1016/S0304-3975(97)00228-4.

3 Simone Bova. Sdds are exponentially more succinct than obdds. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 929–935. AAAI Press, 2016. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12270.

4 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986. doi:10.1109/TC.1986.1676819.

5 Alessio Conte, Roberto Grossi, Andrea Marino, and Luca Versari. Sublinear-space bounded-delay enumeration for massive network analytics: Maximal cliques. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 148:1–148:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.148.

6 William J. Cook and Paul D. Seymour. Tour merging via branch-decomposition. *INFORMS J. Comput.*, 15(3):233–248, 2003. doi:10.1287/ijoc.15.3.233.16078.

7 Gary Hardy, Corinne Lucet, and Nikolaos Limnios. K-terminal network reliability measures with binary decision diagrams. *IEEE Trans. Reliability*, 56(3):506–515, 2007. doi:10.1109/TR.2007.898572.

8 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.

9 Takeru Inoue, Hiroaki Iwashita, Jun Kawahara, and Shin-ichi Minato. Graphillion: software library for very large sets of labeled graphs. *Int. J. Softw. Tools Technol. Transf.*, 18(1):57–66, 2016. doi:10.1007/s10009-014-0352-z.

10 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 Trans. Smart Grid*, 5(1):102–111, 2014. doi:10.1109/TSG.2013.2288976.

11 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.

12 Jun Kawahara, Takashi Horiyama, Keisuke Hotta, and Shin-ichi Minato. Generating all patterns of graph partitions within a disparity bound. In Sheung-Hung Poon, Md. Saidur Rahman, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation, 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings*, volume 10167 of *Lecture Notes in Computer Science*, pages 119–131. Springer, 2017. doi:10.1007/978-3-319-53925-6_10.

13 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*, E100-A(9):1773–1784, 2017.

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

15 Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In Alfred E. Dunlop, editor, *Proceedings of the 30th Design Automation Conference. Dallas, Texas, USA, June 14-18, 1993*, pages 272–277. ACM Press, 1993. doi:10.1145/157485.164890.

16 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*, E101-A(9):1363–1374, 2018.

17 Yu Nakahata, Jun Kawahara, and Shoji Kasahara. Enumerating graph partitions without too small connected components using zero-suppressed binary and ternary decision diagrams. In Gianlorenzo D'Angelo, editor, *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, volume 103 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. `doi:10.4230/LIPIcs.SEA.2018.21`.

18 Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Zero-suppressed sentential decision diagrams. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 1058–1066. AAAI Press, 2016. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12434`.

19 Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Compiling graph substructures into sentential decision diagrams. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 1213–1221. AAAI Press, 2017. URL: `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14919`.

20 Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the tutte polynomial of a graph of moderate size. In John Staples, Peter Eades, Naoki Katoh, and Alistair Moffat, editors, *Algorithms and Computation, 6th International Symposium, ISAAC '95, Cairns, Australia, December 4-6, 1995, Proceedings*, volume 1004 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 1995. `doi:10.1007/BFb0015427`.

21 Akiyoshi Shioura, Akihisa Tamura, and Takeaki Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3):678–692, 1997. `doi:10.1137/S0097539794270881`.

22 Yexiang Xue, Arthur Choi, and Adnan Darwiche. Basing decisions on sentences in decision diagrams. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. URL: `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4977`.