


CompDP: A Framework for Simultaneous Subgraph Counting Under Connectivity Constraints

Kengo Nakamura ✉ 

NTT Communication Science Laboratories, Kyoto, Japan
Graduate School of Informatics, Kyoto University, Japan

Masaaki Nishino ✉ 

NTT Communication Science Laboratories, Kyoto, Japan

Norihito Yasuda ✉

NTT Communication Science Laboratories, Kyoto, Japan

Shin-ichi Minato ✉ 

Graduate School of Informatics, Kyoto University, Japan

Abstract

The subgraph counting problem computes the number of subgraphs of a given graph that satisfy some constraints. Among various constraints imposed on a graph, those regarding the connectivity of vertices, such as “these two vertices must be connected,” have great importance since they are indispensable for determining various graph substructures, e.g., paths, Steiner trees, and rooted spanning forests. In this view, the subgraph counting problem under connectivity constraints is also important because counting such substructures often corresponds to measuring the importance of a vertex in network infrastructures. However, we must solve the subgraph counting problems multiple times to compute such an importance measure for every vertex. Conventionally, they are solved separately by constructing decision diagrams such as BDD and ZDD for each problem. However, even solving a single subgraph counting is a computationally hard task, preventing us from solving it multiple times in a reasonable time. In this paper, we propose a dynamic programming framework that simultaneously counts subgraphs for every vertex by focusing on similar connectivity constraints. Experimental results show that the proposed method solved multiple subgraph counting problems about 10–20 times faster than the existing approach for many problem settings.

2012 ACM Subject Classification Mathematics of computing → Graph theory

Keywords and phrases Subgraph counting, Connectivity, Zero-suppressed Binary Decision Diagram

Digital Object Identifier 10.4230/LIPIcs.SEA.2023.11

Supplementary Material *Software*: <https://github.com/nttcslab/compdp-counting>

Funding This work was supported by JSPS KAKENHI Grant Number JP20H05963 and JST CREST Grant Number JPMJCR22D3.

1 Introduction

Given graph $G = (V, E)$, the *subgraph counting problem* computes the (possibly weighted) count of the subgraphs of G that satisfy some constraints such as each vertex’s degree and the existence of cycles. More specifically, given edge weights $w_e^+, w_e^- \in \mathbb{R}$ for $e \in E$, this problem (exactly) computes the following value:

$$W(\mathcal{E}) := \sum_{E' \in \mathcal{E}} \prod_{e \in E'} w_e^+ \cdot \prod_{e \in E \setminus E'} w_e^-, \quad (1)$$

where $\mathcal{E} \subseteq 2^E$ is a family of the subsets of edges, i.e., *subgraphs*, that satisfy given constraints.



© Kengo Nakamura, Masaaki Nishino, Norihito Yasuda, and Shin-ichi Minato;
licensed under Creative Commons License CC-BY 4.0

21st International Symposium on Experimental Algorithms (SEA 2023).

Editor: Loukas Georgiadis; Article No. 11; pp. 11:1–11:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This problem has been studied as a fundamental task in computer science [2, 1, 5, 4], and extensively studied in the area of *network reliability analysis* [21]. The most fundamental problem of network reliability analysis is computing the probability that the predetermined vertices will remain connected assuming that each edge fails independently with a given probability. This task is equivalent to the subgraph counting problem under *connectivity constraints*; a connectivity constraint is a topological constraint requiring that some pairs of vertices are connected and other pairs are disconnected.

A connectivity constraint is also fundamental in determining various graph substructures, such as paths, Steiner trees, spanning trees, and rooted spanning forests, in combination with other constraints, as described in Section 2. Counting these substructures also has great importance, especially for evaluating the importance of a vertex. For example, paths and Steiner trees on communication networks correspond to the routing of point-to-point and multi-site communication (e.g., see [10, 26]). Thus, the number of paths or Steiner trees passing vertex v is an importance measure for v in this communication network, since its failure causes the loss of this number of communication routing. A cycle passing through source vertex s and another vertex v constitutes a vertex-disjoint two paths between s and v , and counting such cycles corresponds to the number of non-blocking pairs of communication routings from s to v [11]. A rooted spanning forest (RSF) rooted at r_1, \dots, r_k corresponds to a (electrical) distribution network whose substations are located at r_1, \dots, r_k [12]. When we add a new substation to v , the number of RSFs rooted at r_1, \dots, r_k, v (given other constraints such as electric constraints) provides flexibility of the distribution network.

In evaluating such an importance measure for every vertex v , we generally have to solve the subgraph counting problem for every v . That is, we must compute multiple count values $W(\mathcal{E}_{v_1}), \dots, W(\mathcal{E}_{v_n})$ for different families of subgraphs $\mathcal{E}_{v_1}, \dots, \mathcal{E}_{v_n}$. However it was proven that the network reliability evaluation described above is in $\#P$ -complete [32], a computationally challenging class, and computing $W(\mathcal{E})$ in the presence of other constraints is equally as difficult in general. Even a practically fast algorithm for computing $W(\mathcal{E})$ described below may take several minutes or more for a graph with hundreds of edges. Subgraph counting for every vertex described above seems computationally much more difficult since we have to repeatedly solve cumbersome counting tasks.

This paper proposes a practically fast algorithm for *simultaneously* counting subgraphs for every vertex (formally defined in Section 2). Here, “simultaneously” means that we build only one data structure for obtaining all count values $W(\mathcal{E}_{v_1}), \dots, W(\mathcal{E}_{v_n})$. Our contribution is summarized as follows:

- Our proposed method enables us to simultaneously count such graph substructures as paths, cycles, Steiner trees, and RSFs by sophisticated dynamic programming (DP) on the built data structure.
- Complexity analyses show that the proposed method solves subgraph counting for every vertex $O(n)$ times faster than the baseline method that separately solves each counting, where n is the number of vertices.
- We empirically confirmed that the proposed algorithm solved subgraph counting for every vertex around 10–20 times faster than the baseline method.

1.1 Related Works

The study of network reliability problems, i.e., subgraph counting problems under constraints of the form “specified vertices must be connected,” has a long history. This problem is known as $\#P$ -complete [32], meaning that it is computationally tough. Meanwhile, various algorithms have been proposed for this problem, e.g., sum-of-disjoint product [6] with

tieset [25] or cutset enumeration [31], and factoring [33]. However, currently the practically fastest algorithm, originated by Hardy et al. [8] and Herrmann [9], directly constructs a data structure called a *binary decision diagram* (BDD) [3]. Their method successfully computes network reliability for real topologies with around 200 edges. Here BDD is used as a tool that represents the family of subgraphs where predetermined vertices are connected; after it is built, the counting problem can be solved by a simple DP on it.

Other studies exist for counting subgraphs with additional constraints other than connectivity. The simplest method enumerates all the substructures [23, 28] such as paths and spanning trees. Especially, the enumeration of spanning trees [28] corresponds to computing the Tutte polynomial of a graph, which can be used for many kinds of graph counting problems. However, since there might be an exponential number of substructures, enumeration suddenly becomes intractable with the growth of graph size. For practically fast counting, indexing the constrained subgraphs into BDD-like structures has also been studied. Sekine et al. [27] designed an algorithm to build a BDD representing all the spanning trees to compute the Tutte polynomial. Knuth [18] proposed a very efficient scheme called Simpath that indexes all the simple paths in a *zero-suppressed BDD* (ZDD) [20], which is a variant of BDD. By expanding such research, Kawahara et al. [17] proposed a *frontier-based search* (FBS), which can build a ZDD of various graph substructures. Since ZDD also allows a simple DP for counting, FBS can be used for practically fast subgraph counting. Our proposed algorithm is also based on FBS. Subgraph counting is also studied in the context of parameterized complexity theory [5, 4], although their interest often focuses on theoretical aspects. To the best of our knowledge, no works have outperformed the BDD/ZDD-based methods in practically solving subgraph counting problems, including network reliability evaluations.

In 2021, Nakamura et al. [22] proposed an algorithm for network reliability that simultaneously computes the probability of connecting to servers for every client. It essentially solves counting problems for every vertex and runs much faster than the baseline where each client's reliability is computed separately. However, it can only deal with the constraints of the form "all the specified vertices are connected" and cannot accept the constraints of disconnection and others. Technically, the proposed method can deal with these constraints by utilizing the ZDD structures [20] and the FBS [17]. While the existing method [22] relies on a BDD-like structure that is not truly a BDD, we build a legitimate ZDD by FBS, enabling us to combine such existing ZDD algorithms as Apply [20] and subsetting [15].

1.2 Organization of Paper

The rest of this paper is organized as follows. Section 2 describes the preliminary and the formal statement of the problem we solved. Section 3 gives the overview of the proposed method. Section 4 introduces the ZDD and the frontier-based search that are used in the proposed method. Section 5 details the proposed method, and Section 6 analyzes the computational complexity of it. Section 7 empirically compares the proposed method with the baseline in terms of computational time, and Section 8 gives a conclusion.

We give the list of acronyms and notations used in this paper in Table 1.

2 Problem Statement

Before proceeding to our problem statement, we introduce a notion that represents the connectivity constraint in the same manner as Kawahara et al. [17]. As described in Section 1, a connectivity constraint requires that some pairs of vertices are connected and other pairs are disconnected. We represent the connectivity constraint by *subpartition* P

■ **Table 1** Acronyms and notations.

Acronym	
RSF	Rooted Spanning Forest
DP	Dynamic Programming
BDD	Binary Decision Diagram
ZDD	Zero-suppressed binary Decision Diagram
FBS	Frontier-Based Search
DAG	Directed Acyclic Graph
Notation	
<i>Frequently used notations</i>	
$G = (V, E)$	undirected graph with vertex set V and edge set E
n, m	$= V , E $: the number of vertices and edges in the graph
w_e^+, w_e^-	$\in \mathbb{R}$: edge weights for $e \in E$
$W(\mathcal{E})$	the value of subgraph counting given $\mathcal{E} \subseteq 2^E$ (family of subgraphs)
P	subpartition of V representing connectivity constraint
P^*	subpartition of $V \cup \{*\}$ containing one wildcard $*$
$P^*[v]$	connectivity constraint obtained by substituting $*$ in P^* with $v \in V$
$P^*[\]$	connectivity constraint obtained by removing $*$ from P^*
$\mathcal{C}(P)$	$\subseteq 2^E$: family of subgraphs satisfying connectivity constraint P
\mathcal{F}	$\subseteq 2^E$: base set in our problem
$\text{count}(v)$	$= W(\mathcal{F} \cap \mathcal{C}(P^*[v]))$: count value for vertex $v \in V$ computed in our problem
<i>Notations for ZDD</i>	
$Z = (N, A)$	ZDD with node set N and arc set A
\top, \perp	terminal nodes of ZDD
\hat{r}	$\in N$: root node of ZDD
$\text{lo}(\hat{n}), \text{hi}(\hat{n})$	$\in A$: lo-arc and hi-arc outgoing from ZDD node \hat{n}
\hat{n}^-, \hat{n}^+	$\in N$: lo-child and hi-child of ZDD node \hat{n}
$\text{lb}(\hat{n})$	$\in \mathbb{Z}$: label of ZDD node \hat{n}
L_i	i -th level of ZDD, i.e., the set of ZDD nodes whose label is i
R	(directed) path in ZDD
$\mathcal{R}_Z(\hat{n}, \hat{n}')$	set of paths between ZDD nodes \hat{n} and \hat{n}' in ZDD Z
$S(Z)$	$\subseteq 2^E$: family of subgraphs represented by ZDD Z
$E(R)$	$\subseteq E$: subgraph represented by path R in ZDD
$W_p(R)$	$\in \mathbb{R}$: path product of path R in ZDD defined in (3)
$\hat{n}.\mathbf{p}, \hat{n}.\mathbf{r}$	sum of path products of the paths in $\mathcal{R}_Z(\hat{r}, \hat{n})$ and $\mathcal{R}_Z(\hat{n}, \top)$
<i>Notations for explaining existing and proposed methods</i>	
$E_{<i}$	$= \{e_1, \dots, e_{i-1}\}$: processed edges
$E_{\geq i}$	$= \{e_i, \dots, e_m\}$: unprocessed edges
F_i	$\subseteq V$: i -th frontier, the vertices appearing in both $E_{<i}$ and $E_{\geq i}$
V_P	$\subseteq V$: set of vertices appearing in connectivity constraint P
V_{P^*}	$\subseteq V$: set of vertices in the set in P^* containing $*$
V_{P^*}''	$\subseteq V$: set of vertices present in P^* and not included in V_{P^*}'
$\hat{n}.\mathbf{comp}$	partition of F_i maintaining the connectivity among F_i
$\hat{n}.\mathbf{vset}$	connectivity constraint maintaining the connectivity among $F_i \cup V_P$
\mathcal{R}_v	$\subseteq \mathcal{R}_Z(\hat{r}, \top)$: set of paths whose corresponding subgraph satisfies $(\#_v)$
$\mathcal{R}_{v, \hat{n}}$	$\subseteq \mathcal{R}_v$: set of paths that passes through ZDD node \hat{n}
B	$\in \hat{n}.\mathbf{comp}$: set contained in \mathbf{comp} , i.e., connected component
$\mathcal{R}_{\hat{n}, B}$	$\subseteq \mathcal{R}_Z(\hat{n}, \top)$: set of paths associated with ZDD node \hat{n} and set $B \in \hat{n}.\mathbf{comp}$
$\hat{n}.\mathbf{q}[B]$	sum of path products of the paths in $\mathcal{R}_{\hat{n}, B}$
$\hat{n}.\mathbf{q}^-[B]$	sum of path products of the paths in $\mathcal{R}_{\hat{n}, B}$ traversing $\text{lo}(\hat{n})$
$\hat{n}.\mathbf{q}^+[B]$	sum of path products of the paths in $\mathcal{R}_{\hat{n}, B}$ traversing $\text{hi}(\hat{n})$
$V_{\hat{n}.\mathbf{vset}}'$	$\subseteq V$: set of vertices in the set in $\hat{n}.\mathbf{vset}$ containing $*$
$V_{\hat{n}.\mathbf{vset}}''$	$\subseteq V$: set of vertices present in $\hat{n}.\mathbf{vset}$ and not included in $V_{\hat{n}.\mathbf{vset}}'$
$Z_{\mathcal{F}}$	ZDD representing base set \mathcal{F}
<i>Notations for conducting complexity analysis</i>	
$Z_{\text{FBS}(P)}$	ZDD built by FBS with connectivity constraint P
c_P	the number of set in connectivity constraint P
v_P	the number of vertices in connectivity constraint P excluding $*$
\mathbf{fw}	$= \max_i F_i $: frontier width
D_k	k -th Bell number

of vertex set V of graph $G = (V, E)$, where a subpartition of V is a set of pairwise disjoint subsets of V . P imposes the following constraints: (i) for any pair of vertices v, v' in the same set in P , they must be connected, and (ii) for any pair of vertices v, v' in different sets in P , they must not be connected.

We extend the notion by introducing exactly one *wildcard* $*$, which will be replaced by a vertex to represent various connectivity constraints. Let P^* be a subpartition of $V \cup \{*\}$ that must contain exactly one $*$. For $v \in V$, let $P^*[v]$ be the connectivity constraint obtained by substituting $*$ in P^* with v . Additionally, let $P^*[]$ be the connectivity constraint obtained by simply removing $*$ from P . Note that if $v \in V$ is already present in P^* , $P^*[v]$ equals (i) $P^*[]$ if v is in the same set in P^* that contains $*$; or (ii) an inconsistent constraint. For example, for $P^* = \{\{v_1, v_2, *\}, \{v_3\}\}$, $P^*[v_4] = \{\{v_1, v_2, v_4\}, \{v_3\}\}$, $P^*[v_2] = P^*[] = \{\{v_1, v_2\}, \{v_3\}\}$, and $P^*[v_3]$ is an inconsistent constraint.

Now we proceed to our problem definition. In our problem, we are given connected undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, connectivity constraint P^* containing exactly one wildcard $*$, family \mathcal{F} of subgraphs, i.e., subsets of edges, and weights $w_e^+, w_e^- \in \mathbb{R}$ for each $e \in E$. For connectivity constraint P , let $\mathcal{C}(P)$ be the family of subgraphs satisfying P . Our goal is to compute the following value

$$\text{count}(v) := W(\mathcal{F} \cap \mathcal{C}(P^*[v])) \quad \text{for every } v \in V. \quad (2)$$

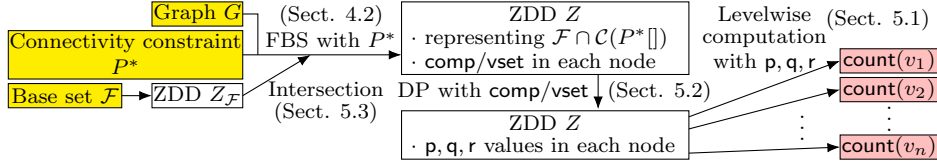
The problems described in Section 1 can be covered by our problem.

- *Path*: Given $s, t \in V$, we set $P^* = \{\{s, t, *\}\}$ and let \mathcal{F} be a family of subgraphs where (i) the degree of s and t is 1, and the others have degree 0 or 2, and (ii) there are no cycles. Then $\text{count}(v)$ equals the number of simple s, t -paths that pass through v .
- *Cycle*: Given $s \in V$, we set $P^* = \{\{s, *\}\}$, and let \mathcal{F} be a family of subgraphs where (i) the degree of each vertex is 0 or 2, and (ii) there is exactly one connected component. Then $\text{count}(v)$ equals the number of cycles starting from s that pass through v .
- *Steiner tree*: Given $T \subseteq V$, we set $P^* = \{T \cup \{*\}\}$ and let \mathcal{F} be a family of subgraphs where there are no cycles and exactly one connected component. Then $\text{count}(v)$ equals the number of T -Steiner trees containing v , where T -Steiner trees are the trees connecting all the T vertices.
- *Rooted spanning forest*: Given $T = \{r_1, \dots, r_k\} \subseteq V$, we set $P^* = \{\{r_1\}, \dots, \{r_k\}, \{*\}\}$, and let \mathcal{F} be a family of subgraphs where (i) every vertex has degree at least 1, (ii) there are no cycles, and (iii) there are exactly $(k + 1)$ connected components. Then $\text{count}(v)$ equals the number of rooted spanning forests rooted at r_1, \dots, r_k and v .

In addition, the problem setting of Nakamura et al. [22] essentially counts the subgraphs where given vertex set $T \subseteq V$ and vertex v are connected for every v and can be recovered by $P^* = \{T \cup \{*\}\}$ and $\mathcal{F} = 2^E$. Although here we list only topological constraints for \mathcal{F} , we can also impose non-topological constraints with \mathcal{F} , such as knapsack constraints.

3 Overview of Proposed Algorithm

First, we explain case $\mathcal{F} = 2^E$. Given connectivity constraint P^* , the baseline method, which separately computes the count value for every v by FBS [17], builds a ZDD with connectivity constraint $P^*[v]$ for every $v \in V$. Here ZDD compactly represents a family of subgraphs by a rooted directed acyclic graph. By FBS with $P^*[v]$, a ZDD representing $\mathcal{C}(P^*[v])$ is built and allows a simple DP for computing $\text{count}(v) = W(\mathcal{C}(P^*[v]))$. The details of ZDD and FBS are explained in Section 4.



■ **Figure 1** Overview of proposed algorithm.

Similarly, the proposed method builds a ZDD by FBS. However, unlike the baseline method, we build only one ZDD Z for P^* , which represents $\mathcal{C}(P^*[\])$. Instead, we retain the information used in the FBS for building Z , `comp` and `vset` in each node of Z , both of which are discarded after the FBS in the baseline method. Since `comp` and `vset` provide rich information for connectivity among vertices, we fully exploit them to perform a more sophisticated DP, yielding for each node of Z three kinds of values, p , q , and r . Their definitions are described in Section 5.1. By using them, we can compute $\text{count}(v)$ values for every $v \in V$. Since the computation of p , q , r , and $\text{count}(v)$ can be performed in time proportional to the execution of FBS, the proposed algorithm runs faster than the baseline. We fully describe the computation of the $\text{count}(v)$ values in Section 5.1 and those of p , q , and r (the DP procedure) in Section 5.2.

Finally, we deal with case $\mathcal{F} \neq 2^E$. For it, we first construct a ZDD $Z_{\mathcal{F}}$ by the existing methods. Then, we construct one ZDD Z that represents $\mathcal{F} \cap \mathcal{C}(P^*[\])$ whose nodes have `comp` and `vset`. This can be performed by exploiting existing techniques of constructing a ZDD of set intersection, such as Apply [20] and subsetting [15]. After Z is built, we can perform the same DP scheme as above. We describe taking the set intersection in Section 5.3. An overview of the proposed method is given in Fig. 1.

By changing base set \mathcal{F} and connectivity constraint P^* , the proposed algorithm can solve various subgraph counting problems, as in Section 2. We named our proposed algorithm *compDP* since it fully uses information `comp`.

3.1 Intuition and Idea behind the Proposed Algorithm

We describe the high-level idea behind the proposed algorithm. As described later, ZDD, which is a rooted and layered directed acyclic graph, represents a family of subgraphs as the set of paths from the root to a terminal node. By defining the *path product* of a path by the weights along this path (precise definition is later), the count value equals the sum of path products of these paths. The intuitive for the proposed algorithm is as follows: Let $\mathcal{E}_v = \mathcal{F} \cap \mathcal{C}(P^*[v])$. To compute $\text{count}(v) = W(\mathcal{E}_v)$ for every $v \in V$, it is sufficient to build a ZDD representing \mathcal{E}_v for every v . However, since \mathcal{E}_v and \mathcal{E}_w ($v \neq w$) are similar families stemming from the common constraint P^* , the ZDDs representing them also are expected to exhibit similar structures. We use such similarities to reduce the computation.

More specifically, we use the following step-by-step ideas: First, since $\mathcal{C}(P^*[v]) \subseteq \mathcal{C}(P^*[\])$ for any $v \in V$, $\mathcal{F} \cap \mathcal{C}(P^*[v])$ is represented by the subset of the paths within the ZDD representing $\mathcal{F} \cap \mathcal{C}(P^*[\])$. Second, these paths can be decomposed into former and latter parts; the former is the paths from the root to a specific layer, and the latter is the paths from the specific layer to the terminal. The count value $\text{count}(v)$ can be represented by the sums of path products of the former part and those of the latter part. Third, when considering such a decomposition for every $v \in V$, we can reuse the values of “the sums of path products of the former part” (p in the proposed algorithm) and “those of the latter part” (q and r in the proposed algorithm). Thus, by pre-computing them by a DP, we can compute $\text{count}(v)$ for every $v \in V$ with these values.

4 ZDD and Frontier-based Search

4.1 Zero-suppressed Binary Decision Diagrams (ZDDs)

Zero-suppressed binary decision diagram (ZDD) [20] $Z = (N, A)$ is a rooted directed acyclic graph (DAG)-shaped data structure representing a family of subsets of edges E ;¹ the root node is denoted by $\hat{r} \in N$. Node set N has two special nodes \top and \perp called *terminals* and other internal nodes. Each internal node \hat{n} has exactly two outgoing arcs called *lo-arc* $\text{lo}(\hat{n})$ and *hi-arc* $\text{hi}(\hat{n})$ and *label* $\text{lb}(\hat{n})$ that is an integer of range $[1, m]$. We respectively call the child nodes of \hat{n} pointed by $\text{lo}(\hat{n})$ and $\text{hi}(\hat{n})$ *lo-child* \hat{n}^- and *hi-child* \hat{n}^+ of \hat{n} . Labels must be ordered in an ascending manner, i.e., $\text{lb}(\hat{n}) < \text{lb}(\hat{n}^-)$ and $\text{lb}(\hat{n}) < \text{lb}(\hat{n}^+)$ must hold for every \hat{n} . For convenience, we set the labels of the terminal nodes to $m + 1$. A ZDD is *normalized* if for every internal node \hat{n} , each of \hat{n}^- and \hat{n}^+ is either \perp or a node whose label is exactly $\text{lb}(\hat{n}) + 1$. Size $|Z|$ of ZDD Z is defined as the number of nodes in Z .

For $\hat{n}, \hat{n}' \in N$, $\mathcal{R}_Z(\hat{n}, \hat{n}')$ denotes the set of paths from \hat{n} to \hat{n}' . Given a predefined order of edges, e_1, \dots, e_m , ZDD Z represents a family of subgraphs $\mathcal{S}(Z) \subseteq 2^E$ by $\mathcal{R}_Z(\hat{r}, \top)$. For path R in Z , we associate subset $E(R) \subseteq E$ where $e_i \in E(R)$ if and only if R traverses a hi-arc outgoing from a node with label i . Then $\mathcal{S}(Z) = \{E(R) \mid R \in \mathcal{R}_Z(\hat{r}, \top)\}$. For example, in Fig. 2(b), path 1-(hi)-2-(lo)-3-(lo)-4-(hi)-5-(hi)- \top indicates that $\{e_1, e_4, e_5\} \in \mathcal{S}(Z)$.

Normalized ZDDs can be used as an efficient tool for subgraph counting. Let Z be a normalized ZDD, and let $R \in \mathcal{R}_Z(\hat{n}, \hat{n}')$ be an arbitrarily chosen path. Given weights $w_e^+, w_e^- \in \mathbb{R}$ for each $e \in E$, we define the *path product* of R by

$$W_p(R) := \prod_{e \in E(R)} w_e^+ \cdot \prod_{e \in \{e_{\text{lb}(\hat{n})}, \dots, e_{\text{lb}(\hat{n}')-1}\} \setminus E(R)} w_e^-. \quad (3)$$

In other words, $W_p(R)$ is the product of w_e^+ for the edges in $E(R)$ and w_e^- for the edges not in $E(R)$. We also define value $\hat{n}.\mathbf{p}$ for ZDD node \hat{n} by the sum of path products of the paths in $\mathcal{R}_Z(\hat{r}, \hat{n})$. By definition, $W(\mathcal{S}(Z))$ equals $\top.\mathbf{p}$. Moreover, although $\hat{n}.\mathbf{p}$ is defined as the sum of a possibly exponential number of path products, its value can efficiently be computed by a DP. Simple calculation shows the following equation for a node other than \hat{r} or \perp :

$$\hat{n}.\mathbf{p} = \sum_{\hat{n}':(\hat{n}')^- = \hat{n}} w_{e_{\text{lb}(\hat{n})}}^- \cdot \hat{n}'.\mathbf{p} + \sum_{\hat{n}':(\hat{n}')^+ = \hat{n}} w_{e_{\text{lb}(\hat{n})}}^+ \cdot \hat{n}'.\mathbf{p}. \quad (4)$$

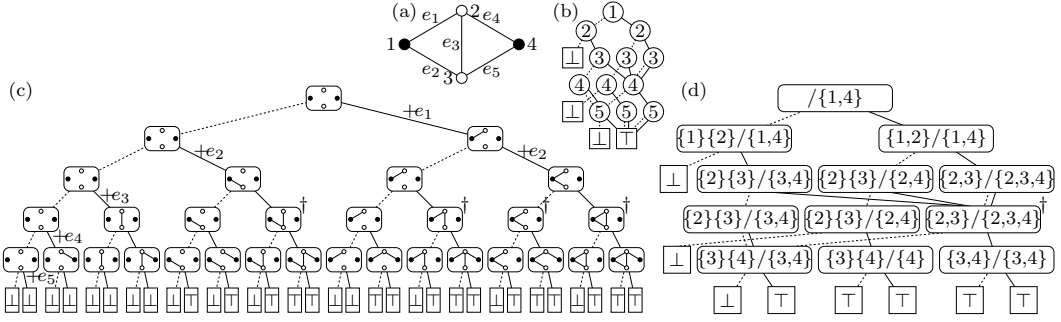
Starting with $\hat{r}.\mathbf{p} = 1$, by applying (4) in a top-down manner along Z , we can compute the value of $\top.\mathbf{p} = W(\mathcal{S}(Z))$ in time proportional to the number of nodes in Z .

4.2 Frontier-based Search

A frontier-based search (FBS) [17] is an efficient method for constructing a normalized ZDD that represents a family of subgraphs satisfying some constraints. Below we explain its procedure of given connectivity constraint P (without $*$).

We start with a naive way for constructing ZDD. Given order of edges e_1, \dots, e_m , we decide one by one whether e_i is excluded or included in the subgraph. This generates a binary decision tree like Fig. 2(c). After deciding every link's exclusion or inclusion, all subgraphs are enumerated at the bottom of the decision tree, and we can judge whether the constraints

¹ Although a ZDD can represent a family of subsets of arbitrary base set X , here we explain it as a tool for representing a family of edge subsets for simplicity.



■ **Figure 2** (a) Example of graph. (b) Example of (normalized) ZDD. Dashed and solid lines indicate lo- and hi-arcs, and an integer inside a node represents a label. (c) Binary decision tree made by deciding one by one whether e_i is excluded or included. (d) ZDD made by FBS where $P = \{\{1, 4\}\}$. Two subpartitions of vertices inside a node represent **comp** and **vset**.

are satisfied one by one. The example in Fig. 2(c) judges whether the black vertices (1 and 4) are connected. This decision tree has a property where the collection of paths from root to \top corresponds to the family of subgraphs satisfying the constraints. After making the decision tree, ZDD can be constructed by merging the identical subtrees of it. For example, the four nodes marked \dagger in Fig. 2(c) have an identical pattern of leaves: \perp, \top, \top, \top . Even if we merge them, the property that the collection of paths from root to \top corresponds to a family of satisfying subgraphs is not broken.

However, since there are 2^m subgraphs, the size of the decision tree must be exponential. For efficient construction of ZDD, we try to detect identical subtrees without constructing them. Formally, the identicalness of subtrees can be stated as follows. Let $E_{<i} = \{e_1, \dots, e_{i-1}\}$ and $E_{\geq i} = \{e_i, \dots, e_m\}$. We call the subset of edges after deciding e_{i-1} 's exclusion or inclusion the i -th subgraph. Note that the i -th subgraphs are subsets of $E_{<i}$. We consider the following equivalence relation for i -th subgraphs $X, X' \subseteq E_{<i}$: (§) For any $Y \subseteq E_{\geq i}$, whether subgraph $X \cup Y$ or $X' \cup Y$ satisfies the constraints is equivalent. If (§) holds, the subtree rooted at X and that rooted at X' are identical.

Condition (§) is met for $X, X' \subseteq E_{<i}$ if X and X' share identical connectivity among V , i.e., for any $v, v' \in V$, whether v and v' are connected is equivalent. This is because if X and X' share identical connectivity, so do $X \cup Y$ and $X' \cup Y$. Moreover, if X and X' have an identical connectivity, so do $X \cup \{e_i\}$ and $X' \cup \{e_i\}$. This enables us to build a decision “diagram” by the following procedure. In building a decision tree like Fig. 2(c) in a breadth-first manner, we merge some subgraphs if they exhibit an identical connectivity.

The FBS further refines this idea by focusing on the connectivity among a limited subset of vertices. Let F_i , called i -th frontier, be the vertices appearing in both $E_{<i}$ and $E_{\geq i}$. Also, let V_P be the set of vertices appearing in connectivity constraint P . Then, it can be proved that condition (§) is satisfied if X and X' have an identical connectivity among $F_i \cup V_P$. Intuitively, this is because every vertex $v \in V \setminus V_P$ that does not appear in $E_{\geq i}$ can be equated with vertex $v' \in F_i \cup V_P$ if v is connected to v' , or it can be ignored if it is not connected to any vertex in $F_i \cup V_P$. FBS generates a diagram by a breadth-first manner such that the nodes exhibiting identical connectivity among $F_i \cup V_P$ are all merged.

Algorithm 1 is pseudocode of FBS. The connectivity among $F_i \cup V_P$ is maintained by two subpartitions of vertices: **comp** and **vset**. Here **comp** maintains the connectivity among F_i while **vset** maintains the connectivity among the sets in **comp** and the vertices in V_P . More specifically, **comp** is a partition of F_i such that $v, v' \in F_i$ are in the same set if and only if

■ **Algorithm 1** Frontier-based search for connectivity constraint P . Underlined part in line 16 is added if it is used for a subroutine of proposed method.

```

1  $\hat{r}.comp \leftarrow \{\}, \hat{r}.vset \leftarrow P, L_1 \leftarrow \{\hat{r}\}, L_i \leftarrow \emptyset (i = 2, \dots, m + 1)$ 
2 for  $i \leftarrow 1$  to  $m$  do //  $e_i = \{v, v'\}$ 
3   foreach  $\hat{n} \in L_i$  do
4     foreach  $f \in \{-, +\}$  do
5        $\hat{n}' \leftarrow \hat{n}$  //  $V_{\hat{n}', vset}$  is the vertices present in  $\hat{n}'.vset$ 
6       if  $f = +$  and  $v, v' \in V_{\hat{n}', vset}$  and  $\hat{n}'.vset[v] \neq \hat{n}'.vset[v']$  then
7          $\hat{n}' \leftarrow \perp$  and goto finish //  $v$  and  $v'$  must not be connected
8       foreach  $u \in \{v, v'\} \setminus F_i$  do // Vertices entering the frontier
9          $\hat{n}'.comp \leftarrow \hat{n}'.comp \cup \{u\}$  // Add  $u$  as an isolated vertex
10      if  $f = +$  and  $\hat{n}'.comp[v] \neq \hat{n}'.comp[v']$  then // Connecting two components
11        Merge  $\hat{n}'.comp[v]$  and  $\hat{n}'.comp[v']$  into one
12        if  $v \in V_{\hat{n}', vset}$  and  $v' \notin V_{\hat{n}', vset}$  then Add vertices in  $\hat{n}'.comp[v']$  to  $\hat{n}'.vset[v]$ 
13        if  $v \notin V_{\hat{n}', vset}$  and  $v' \in V_{\hat{n}', vset}$  then Add vertices in  $\hat{n}'.comp[v]$  to  $\hat{n}'.vset[v']$ 
14      foreach  $u \in \{v, v'\} \setminus F_{i+1}$  do // Vertices leaving from frontier
15        if  $\{u\} \in \hat{n}'.comp$  then // Component containing  $u$  leaves frontier
16          if  $u \in V_{\hat{n}', vset}$  and  $\{u\} \notin \hat{n}'.vset$  and  $\{u, *\} \notin \hat{n}'.vset$  then
17             $\hat{n}' \leftarrow \perp$  and goto finish //  $u$  must be connected to  $w \in \hat{n}'.vset[u]$ 
18          Remove  $u$  from  $\hat{n}'.comp$  and  $\hat{n}'.vset$  if exists
19      if  $\hat{n}' \neq \perp$  then
20        if  $i = m$  then  $\hat{n}' \leftarrow \top$  // All conditions are satisfied
21        else if  $\exists \hat{n}'' \in L_{i+1}$  s.t.  $\hat{n}'' .comp = \hat{n}'.comp$  and  $\hat{n}'' .vset = \hat{n}'.vset$  then
22           $\hat{n}' \leftarrow \hat{n}''$  // Already generated node
23        else  $L_{i+1} \leftarrow L_{i+1} \cup \{\hat{n}'\}$  // Newly generated node
24      finish:  $\hat{n}^f \leftarrow \hat{n}'$  // Set lo- or hi-child of  $\hat{n}$  to  $\hat{n}'$ 

```

they are connected, and `vset` maintains the connectivity constraint such that the vertices in the same set must be connected and those in different sets must be disconnected. Here `comp[v]` denotes the set in `comp` containing v , and as is the same with `vset[v]`. By starting with `comp` = { } and `vset` = P (Line 1), the algorithm repeatedly updates `comp` and `vset` by excluding ($f = -$) or including ($f = +$) e_i ; lines 5–13 are the update procedure. We set the destination of lo-arc ($f = -$) or hi-arc ($f = +$) to the node with the updated `comp` and `vset` (line 24). If the node having identical `comp` and `vset` has already been generated, we set it to the already generated node (lines 21–22); otherwise, we newly generate a node (line 23). When either of the following occurs, we *prune* the node, i.e., setting the destination of an arc to \perp : (I) Two vertices are connected that are in different sets of `vset` (lines 6–7). This violates the disconnection requirement of `vset`. (II) `comp[v]` leaves the frontier, but `vset[v]` has vertex v' other than v (lines 16–17). In this case v will never be connected with v' , violating the connection requirement of `vset`. If the search proceeds to the final level without being pruned, it reaches \top , i.e., constraint P is satisfied (line 20).

For example, Fig. 2(d) is the result of FBS given the graph in Fig. 2(a) and the constraint that vertices 1 and 4 must be connected. We now focus on the left, 2nd level node: “{1}{2}/{1,4}”. Here `comp` = {1}{2} denotes two components, the one including 1 and the one including 2, and `vset` = {1, 4} represents that 1 and 4 must be connected. If we exclude $e_2 = \{1, 3\}$, the component including 1 is left isolated because $F_3 = \{2, 3\}$ does not include 1. However, since this contradicts that 1 and 4 must be connected, the lo-child is \perp (pruned). If we include e_2 , the component including 1 becomes one that includes 3 at the next level. The constraint that 1 and 4 must be connected can be rewritten as that 3 and 4 must be connected. Thus, hi-child’s `comp` is {2}{3} and `vset` is {3,4}. Here the four nodes marked \dagger in Fig. 2(c) are treated as only one marked node in Fig. 2(d).

5 Details of Proposed Method

First, we assume $\mathcal{F} = 2^E$; this assumption is removed in Section 5.3. As in Section 3, the proposed method first builds ZDD Z representing $\mathcal{C}(P^*[\cdot])$. To explain the meaning and procedure for this, we observe the relationship between $P^*[\cdot]$ and $P^*[v]$. Let V'_{P^*} be the vertices in the set in P^* containing $*$, and let V''_{P^*} be the other vertices present in P^* . From the definition, $P^*[v]$ imposes the following additional constraint on $P^*[\cdot]$:

($\#_v$) v must be connected with the vertices in V'_{P^*} , and v must be disconnected from the vertices in V''_{P^*} .

That is, $\mathcal{C}(P^*[v]) = \{X \mid X \in \mathcal{C}(P^*[\cdot]), X \text{ satisfies } (\#_v)\}$. Now the constraint $P^*[v]$ is decomposed into $(\#_v)$ and $P^*[\cdot]$, where $(\#_v)$ involves only the connectivity around v and $P^*[\cdot]$ represents the other constraints. The fact that $(\#_v)$ concerns only the connectivity around v enables us to compute $\text{count}(v)$ for every v with only one ZDD Z representing $\mathcal{C}(P^*[\cdot])$, as described in the subsequent sections.

Meanwhile, during the procedure of FBS, we must remember which set in `vset` corresponds to the set in P containing $*$ since we use it for the subsequent computation. To achieve this, we just consider $*$ in P^* a special vertex. More specifically, we let $\hat{r}.\text{vset} \leftarrow P^*$ in line 1 of Algorithm 1 and add the underlined part of line 16. By adding the underlined part, the pruning condition (II) simply discards $*$ even if `vset[v]` contains $*$. Thus, Z finally represents $\mathcal{C}(P^*[\cdot])$, while each `vset` has at most one set containing $*$.

5.1 Computation with Intermediate Level of Diagram

Let \mathcal{R}_v be the paths in $\mathcal{R}_Z(\hat{r}, \top)$ whose corresponding subgraph satisfies $(\#_v)$. As stated above, $\text{count}(v)$ equals the sum of path products of the paths in \mathcal{R}_v . Here we focus on i -th level L_i of Z where $v \in F_i$.² For node $\hat{n} \in L_i$ with label i , let $\mathcal{R}_{v,\hat{n}}$ be the paths in \mathcal{R}_v passing through \hat{n} . Since Z is normalized, every $\hat{r}\text{-}\top$ path in Z passes exactly one node in L_i . This means that $\text{count}(v)$ can be represented as the sum of $\sum_{R \in \mathcal{R}_{v,\hat{n}}} W_p(R)$ over $\hat{n} \in L_i$.

We further decompose $\sum_{R \in \mathcal{R}_{v,\hat{n}}} W_p(R)$ by focusing on $\hat{n} \in L_i$. Since $\hat{n}.\text{comp}$ maintains the connectivity among F_i , the sets in $\hat{n}.\text{comp}$ are indeed connected components. Since the connectivity around v can be translated into that around connected component $B = \hat{n}.\text{comp}[v]$, $(\#_v)$ can be restated as a constraint on $B = \hat{n}.\text{comp}[v]$:

($\#'_B$) Connected component B must be connected with the vertices in V'_{P^*} , and B must be disconnected from the vertices in V''_{P^*} .

Let $\mathcal{R}_{\hat{n},B} \subseteq \mathcal{R}_Z(\hat{n}, \top)$ be a set of paths such that $R' \in \mathcal{R}_{\hat{n},B}$ if and only if $E(R) \cup E(R')$ satisfies $(\#'_B)$ for arbitrarily chosen $R \in \mathcal{R}_Z(\hat{r}, \hat{n})$. $\mathcal{R}_{\hat{n},B}$ is well-defined, i.e., kept unchanged regardless of the choice of R because $E(R)$'s connectivity among components and the vertices in $V'_{P^*} \cup V''_{P^*}$ is completely determined in $\hat{n}.\text{vset}$. This means that $\mathcal{R}_{v,\hat{n}}$ can be written as *direct product* $\mathcal{R}_Z(\hat{r}, \hat{n}) \sqcup \mathcal{R}_{\hat{n},\hat{n}.\text{comp}[v]}$ where $A \sqcup B := \{a \cup b \mid a \in A, b \in B\}$. In other words, every path $R \in \mathcal{R}_{v,\hat{n}}$ can be decomposed into $R' \in \mathcal{R}_Z(\hat{r}, \hat{n})$ and $R'' \in \mathcal{R}_{\hat{n},\hat{n}.\text{comp}[v]}$. From the definition of path product, $W_p(R) = W_p(R')W_p(R'')$. Thus, by defining $\hat{n}.\text{q}[B]$ as the sum of path product of the paths in $\mathcal{R}_{\hat{n},B}$, the following holds:

$$\sum_{R \in \mathcal{R}_{v,\hat{n}}} W_p(R) = \sum_{R' \in \mathcal{R}_Z(\hat{r}, \hat{n})} \sum_{R'' \in \mathcal{R}_{\hat{n},\hat{n}.\text{comp}[v]}} W_p(R')W_p(R'') = \hat{n}.\text{p} \cdot \hat{n}.\text{q}[\hat{n}.\text{comp}[v]]. \quad (5)$$

² If v 's degree is more than 1, there is at least one i such that $v \in F_i$. For example, if e_i is the first edge containing v within the edge ordering, $v \in F_i$. The treatment of degree 1 vertices is in Appendix A.1.

Finally, $\text{count}(v)$ can be represented as

$$\text{count}(v) = \sum_{\hat{n} \in L_i} \sum_{R \in \mathcal{R}_{v, \hat{n}}} W_p(R) = \sum_{\hat{n} \in L_i} \hat{n}.p \cdot \hat{n}.q[\hat{n}.\text{comp}[v]]. \quad (6)$$

By choosing i such that $v \in F_i$ for every v , we can compute $\text{count}(v)$ for every v by (6) if p and q are computed. In the next section, we show that q can easily be computed by DP.

5.2 Dynamic Programming

First, we define a correspondence of the components in comp between \hat{n} and its child nodes.

► **Definition 1.** Let $\hat{n} \in L_i$ be a node of a normalized ZDD whose label is i , and let f be either $-$ or $+$. Assuming $\hat{n}^f \neq \perp$, for $B \in \hat{n}.\text{comp}$, we define B^f as follows: (i) If B contains vertex v in F_{i+1} , $B^f = \hat{n}^f.\text{comp}[v]$. (ii) If no such vertex exists, $B^- = \emptyset$, i.e., no corresponding component. For $f = +$, let v, v' be the endpoints of e_i . If $v' \in F_i$ and $v \in B$, $B^+ = \hat{n}^+.\text{comp}[v']$. If $v \in F_i$ and $v' \in B$, $B^+ = \hat{n}^+.\text{comp}[v]$. Otherwise, $B^+ = \emptyset$.

Intuitively, B^f is a component in $\hat{n}^f.\text{comp}$ that represents the same component as B .

We derive a formula for $\hat{n}.q[B]$ by decomposing the set of paths $\mathcal{R}_{\hat{n}, B}$. Since every path in $\mathcal{R}_{\hat{n}, B}$ passes either $\text{lo}(\hat{n})$ or $\text{hi}(\hat{n})$, we have a case analysis. Let $\hat{n}.q^-[B]$ ($\hat{n}.q^+[B]$) be the sum of path products of the paths in $\mathcal{R}_{\hat{n}, B}$ that traverse $\text{lo}(\hat{n})$ ($\text{hi}(\hat{n})$). Now

$$\hat{n}.q[B] = \hat{n}.q^-[B] + \hat{n}.q^+[B]. \quad (7)$$

We now focus on $\hat{n}.q^-[B]$, which means that e_i is excluded. If $\hat{n}^- = \perp$, constraint $\mathcal{C}(P^*[])$ is not satisfied, so no path in $\mathcal{R}_{\hat{n}, B}$ passes through $\text{lo}(\hat{n})$. Thus, $\hat{n}.q^-[B] = 0$. Otherwise, B^- is defined as in Definition 1. If $B^- \neq \emptyset$, since B^- is the same component as B , constraint $(\#'_B)$ is satisfied if and only if constraint $(\#'_{B^-})$ for \hat{n}^- is satisfied. Thus, the set of paths in $\mathcal{R}_{\hat{n}, B}$ that traverse $\text{lo}(\hat{n})$ can be written as $\{\text{lo}(\hat{n})\} \sqcup \mathcal{R}_{\hat{n}^-, B^-}$. This means that $\hat{n}.q^-[B] = w_{e_i}^- \cdot \hat{n}^- .q[B^-]$.

The remaining case is $\hat{n}^- \neq \perp$ and $B^- = \emptyset$. In this case, the component B does not exist in the next level L_{i+1} and thus we cannot translate constraint $(\#'_B)$ into the one concerning the lo-child \hat{n}^- . In other words, we must judge whether constraint $(\#'_B)$ is satisfied with only the information on \hat{n} . Fortunately, it is possible because $\hat{n}.\text{vset}$ completely determines the connectivity among $B \in \hat{n}.\text{comp}$ and $V'_{P^*} \cup V''_{P^*}$.

We have case analysis on how B is connected with $V'_{P^*} \cup V''_{P^*}$; how to distinguish these cases are described later. If B is connected with some (but not all) vertices in V'_{P^*} , it violates the constraint $P^*[]$ that all the vertices in V'_{P^*} are connected. If B is connected with both the vertices in V'_{P^*} and those in V''_{P^*} , it again violates the constraint $P^*[]$ that the vertices in the different sets of $P^*[]$ are disconnected. Therefore, since at least $P^*[]$ is not violated by $R' \in \mathcal{R}_Z(\hat{n}, \hat{n})$, only one of the three cases must hold: (i) B is disconnected from any vertex in P^* , (ii) B is connected with all the vertices in V'_{P^*} , and (iii) B is connected with some vertices in V''_{P^*} . When $V'_{P^*} \neq \emptyset$, only case (ii) satisfies $(\#'_B)$. When $V'_{P^*} = \emptyset$, case (i) also satisfies $(\#'_B)$. For both scenarios, the set of paths in $\mathcal{R}_{\hat{n}, B}$ that traverse $\text{lo}(\hat{n})$ can be written as $\{\text{lo}(\hat{n})\} \sqcup \mathcal{R}_Z(\hat{n}^-, \top)$, since $(\#'_B)$ is always satisfied regardless of the choice of the path from \hat{n}^- to \top . By defining $\hat{n}'.r$ as the sum of path products of the paths in $\mathcal{R}_Z(\hat{n}, \top)$ for any node \hat{n}' , $\hat{n}.q^-[B] = w_{e_i}^- \cdot \hat{n}^- .r$ for these cases. To sum up, the following holds:

■ **Algorithm 2** CompDP: dynamic programming with information of comp.

```

1  $\hat{r}.p \leftarrow 1$ , set all other  $p$  values to 0,  $\top.r \leftarrow 1$ ,  $\perp.r \leftarrow 0$ , set all  $q$  values to 0
2 for  $i \leftarrow 1$  to  $m$  do // Top-down DP
3   foreach  $\hat{n} \in L_i$  do
4     if  $\hat{n}^- \neq \perp$  then  $\hat{n}^-.p += w_{e_i}^- \cdot \hat{n}.p$  // (4)
5     if  $\hat{n}^+ \neq \perp$  then  $\hat{n}^+.p += w_{e_i}^+ \cdot \hat{n}.p$ 
6 for  $i \leftarrow m$  to 1 do // Bottom-up DP
7   foreach  $\hat{n} \in L_i$  do
8      $\hat{n}.r \leftarrow w_{e_i}^- \cdot \hat{n}^-.r + w_{e_i}^+ \cdot \hat{n}^+.r$  // (9)
9     foreach  $B \in \hat{n}.comp$  do
10    foreach  $f \in \{-, +\}$  do
11    if  $\hat{n}^f \neq \perp$  and  $B^f \neq \emptyset$  then  $\hat{n}.q^f[B] \leftarrow w_{e_i}^f \cdot \hat{n}^f.q[B^f]$  // (8), 1st case
12    else if  $\hat{n}^f \neq \perp$  and  $(B \cap V'_{\hat{n}.vset} \neq \emptyset$  or  $(V'_{P^*} = \emptyset$  and  $B \cap V''_{\hat{n}.vset} = \emptyset)$  then
13    [  $\hat{n}.q^f[B] \leftarrow w_{e_i}^f \cdot \hat{n}^f.r$  // (8), 2nd case
14    ] Process corner cases

```

$$\hat{n}.q^-[B] = \begin{cases} w_{e_{\text{lb}(\hat{n})}}^- \cdot \hat{n}^-.q[B^-] & (\hat{n}^- \neq \perp, B^- \neq \emptyset) \\ w_{e_{\text{lb}(\hat{n})}}^- \cdot \hat{n}^-.r & (\hat{n}^- \neq \perp, B^- = \emptyset, \text{case (ii) or (case (i) and } V'_{P^*} = \emptyset)) , \\ 0 & (\text{otherwise}) \end{cases} \quad (8)$$

$$\text{where } \top.r = 1, \perp.r = 0, \hat{n}.r = w_{e_{\text{lb}(\hat{n})}}^- \cdot \hat{n}^-.r + w_{e_{\text{lb}(\hat{n})}}^+ \cdot \hat{n}^+.r. \quad (9)$$

Note that the recurrence formula (9) for r can easily be derived from the definition in the same way as the formula (4) for p .

The remaining is how to distinguish the cases (i)–(iii). Since the connectivity among B and the vertices in P^* is stored in $\hat{n}.vset$, it can be achieved by the comparison of B and $\hat{n}.vset$. Let $V'_{\hat{n}.vset}$ be the vertices in the set in $\hat{n}.vset$ containing $*$ and let $V''_{\hat{n}.vset}$ be the other vertices present in $\hat{n}.vset$. Then, case (i) holds when the vertices in B do not exist in $\hat{n}.vset$. Case (ii) holds when B has a node in $V'_{\hat{n}.vset}$. Case (iii) holds when B has a node in $V''_{\hat{n}.vset}$. Let us see the example by Fig. 2(d). If we perform FBS with $P^* = \{\{1, 4, *\}\}$, the center node of 5th level, say \hat{n} , becomes $\{3\}\{4\}/\{4, *\}$. When traversing $\text{lo}(\hat{n})$, both $\{3\}$ and $\{4\}$ leave from the frontier. Here $\{3\}$ falls into case (i) and $\{4\}$ falls into case (ii), thus we have $\hat{n}.q^-[\{3\}] = 0$ and $\hat{n}.q^-[\{4\}] = w_{e_5}^- \cdot \hat{n}^-.r = w_{e_5}^- \cdot \top.r = w_{e_5}^-$.

Equation (8) also holds even if we substitute $-$ with $+$, which means e_i is included, except for the following corner case. Let $e_i = \{v, v'\}$. We consider the case where $v, v' \notin F_{i+1}$ and $\hat{n}.comp[v]$ leaves the frontier with case (i). When $V'_{P^*} \neq \emptyset$, if $v' \in V'_{\hat{n}.vset}$ or $\hat{n}.comp[v']$ leaves the frontier with case (ii), $\hat{n}.q^+[\hat{n}.comp[v]] = w_{e_i}^+ \cdot \hat{n}^+.r$ since $\hat{n}.comp[v]$ is finally connected with V'_{P^*} . Similarly, when $V'_{P^*} = \emptyset$, if $v' \in V''_{\hat{n}.vset}$ or $\hat{n}.comp[v']$ leaves the frontier with case (iii), $\hat{n}.q^+[\hat{n}.comp[v]] = 0$ since $\hat{n}.comp[v]$ is finally connected with V''_{P^*} .

Algorithm 2 describes the pseudocode for DP. After p , q and r values are computed by Algorithm 2, the $\text{count}(v)$ value for every v can be obtained by (6).

5.3 Intersection with Base Set

Next we generalize for case $\mathcal{F} \neq 2^E$. In the previous sections, we used the fact that $P^*[v]$ is a constraint made by adding another constraint ($\#_v$) to $P^*[]$. This also holds even if \mathcal{F} is constrained, i.e., $\mathcal{F} \cap \mathcal{C}(P^*[v]) = \{X \mid X \in \mathcal{F} \cap \mathcal{C}(P^*[]), X \text{ satisfies } (\#_v)\}$. Therefore, by constructing a ZDD Z representing $\mathcal{F} \cap \mathcal{C}(P^*[])$ with the information of comp and vset , we can reuse the discussions in Sections 5.1 and 5.2 and run Algorithm 2 on Z to obtain $\text{count}(v)$

for every $v \in V$. More specifically, let Z' be the normalized ZDD of $\mathcal{C}(P^*[\])$ built by FBS with P^* . Then Z should be a normalized ZDD representing $\mathcal{F} \cap \mathcal{C}(P^*[\])$ that satisfies the following condition for every i : for any i -th subgraph $X \subseteq E_{<i}$, if X corresponds to i -th level nodes \hat{n} in Z and \hat{n}' in Z' , \hat{n} must have the same **comp** and **vset** as \hat{n}' .

After building $Z_{\mathcal{F}}$ that represents \mathcal{F} by some means, Z can be built by combining FBS with the existing methods. One approach is to use Apply [20]. First, we build Z' representing $\mathcal{C}(P^*[\])$ by FBS. Then by taking the set intersection of $Z_{\mathcal{F}}$ and Z' with Apply while keeping the information of **comp** and **vset**, we can construct Z . The other is to use subsetting [15]. It enables us to directly construct Z from $Z_{\mathcal{F}}$ in a similar manner as the FBS. For the sake of completeness, we describe the pseudocode of subsetting with FBS in Appendix A.2.

6 Complexity Analysis

We here conduct a complexity analysis of the proposed algorithm. For connectivity constraint P possibly including $*$, let $Z_{\text{FBS}(P)}$ be a ZDD built by FBS with P , let c_P be the number of sets in P , and let v_P be the number of vertices in P (excluding $*$). Additionally, let $\text{fw} = \max_i |F_i|$ called *frontier width*. We give detailed proofs in Appendix A.3.

First, we bound the running time of our algorithm by the ZDD size.

► **Proposition 2.** *Our proposed algorithm runs in $O(\text{fw} \cdot |Z_{\mathcal{F}}| |Z_{\text{FBS}(P^*)}|)$ time.*

Next we bound the ZDD sizes. The bound of $|Z_{\text{FBS}(P)}|$ for P excluding $*$ is given in Proposition 3 and that of $|Z_{\text{FBS}(P^*)}|$ for P^* including $*$ is in Proposition 4.

► **Proposition 3.** *The size of $Z_{\text{FBS}(P)}$ for connectivity constraint P excluding $*$ is bounded by $O(mD_{\text{fw}} \cdot \min\{(c_P + 1)^{\text{fw}}, (\text{fw} + 1)^{v_P}\})$, where D_{fw} is the fw -th Bell number.*

► **Proposition 4.** *For connectivity constraint P^* including $*$, $|Z_{\text{FBS}(P^*)}| \leq c_{P^*} |Z_{\text{FBS}(P^*[\])}|$.*

Combining Propositions 2–4 yields the following theorem.

► **Theorem 5.** *The proposed algorithm runs in $O(\text{fw} \cdot c_{P^*} |Z_{\mathcal{F}}| |Z_{\text{FBS}(P^*[\])}|)$ time, which is bounded by $O(|Z_{\mathcal{F}}| \cdot mc_{P^*} \cdot \text{fw} \cdot D_{\text{fw}} \cdot \min\{(c_{P^*} + 1)^{\text{fw}}, (\text{fw} + 1)^{v_{P^*}}\})$.*

If fw can be considered as a constant, the proposed algorithm runs in $O(mc_{P^*} |Z_{\mathcal{F}}|)$ time. It is known that fw is closely related to the *pathwidth* [24] of a graph. If a graph's pathwidth is pw , there is a edge ordering with $\text{fw} = \text{pw}$ [13]. The value of pw is often much smaller than n and m for sparse graphs, e.g., [22]. Although obtaining such an order is NP-hard in general, we can use pathwidth optimization heuristics [13] for obtaining better ordering.

We compare this complexity with the baseline method where we separately build a ZDD representing $\mathcal{F} \cap \mathcal{C}(P^*[v])$ by FBS. The overall complexity is $O(\text{fw} \cdot \sum_v |Z_{\mathcal{F}}| |Z_{\text{FBS}(P^*[v])}|)$, analyzed in the same way as Proposition 2, which is bounded by $O(|Z_{\mathcal{F}}| \cdot mn \cdot \text{fw} \cdot D_{\text{fw}} \cdot \min\{(c_{P^*} + 1)^{\text{fw}}, (\text{fw} + 1)^{v_{P^*}+1}\})$. If fw is constant, it is $O(|Z_{\mathcal{F}}| mn)$. Compared with Theorem 5, the proposed method runs faster by an $O(n)$ factor.

Here we mention the ZDD sizes. The complexity bounds of the proposed and baseline methods heavily depend on $Z_{\mathcal{F}}$'s size. Here $|Z_{\mathcal{F}}|$ also remains small for various constraints if fw is small. For example, the constraints appeared in the example of Section 2, e.g., the degree constraints and the existence of cycles, can all be represented as a ZDD whose size is proportional to m if fw is constant [27, 18, 17]. This boosts the effectiveness of both the proposed and baseline methods for practical use because fw is often much smaller than n and m for graphs in real worlds. Moreover, $|Z|$ is often much smaller than expected from the above analysis, as demonstrated by Kawahara et al. [17].

■ **Table 2** Computational time for grid graphs and Rocketfuel dataset in seconds.

Instance	n	m	fw	Path		Cycle		Steiner tree		RSF	
				Ours	Base	Ours	Base	Ours	Base	Ours	Base
Grid-8x8	64	112	8	0.06	0.48	0.06	0.54	4.93	29.87	8.87	38.17
Grid-8x16	128	232	8	0.16	2.54	0.16	2.81	14.16	183.16	25.72	234.17
Grid-8x24	192	352	8	0.26	6.29	0.27	6.86	23.27	470.60	42.81	>600
Grid-8x32	256	472	8	0.36	11.73	0.38	12.65	32.55	>600	60.28	>600
Grid-9x9	81	144	9	0.22	2.13	0.22	2.34	40.13	298.57	62.44	397.57
Grid-10x10	100	180	10	0.78	9.70	0.89	11.69	284.17	>600	430.04	>600
Grid-11x11	121	220	11	2.88	42.26	3.22	50.97	>600	>600	>600	>600
Grid-12x12	144	264	12	11.24	183.87	15.25	241.94	>600	>600	>600	>600
Grid-13x13	169	312	13	45.51	>600	56.25	>600	>600	>600	>600	>600
Rocketfuel-1221	318	758	10	157.01	>600	111.52	>600	181.38	>600	>600	>600
Rocketfuel-1755	172	381	12	43.94	>600	32.48	>600	>600	>600	>600	>600
Rocketfuel-6461	182	294	10	3.66	65.64	4.80	128.86	71.10	>600	95.47	>600

We close this section by mentioning the space complexity of the proposed and the baseline methods. The proposed algorithm uses at most $O(\text{fw} \cdot |Z_{\mathcal{F}}| |Z_{\text{FBS}(P^*)}|)$ words of space, since it retains $O(\text{fw})$ words of information for each node of $Z_{\text{FBS}(P^*)}$. The baseline method typically uses at most $O(\text{fw} \cdot |Z_{\mathcal{F}}| |Z_{\text{FBS}(P^*[v])}|)$ words of space for the computation of $\text{count}(v)$. If it is assumed that $|Z_{\text{FBS}(P^*[v])}|$ is close to $|Z_{\text{FBS}(P^*)}|$, the space complexity of the baseline method is at most only $O(c_{P^*})$ times smaller due to Proposition 4.

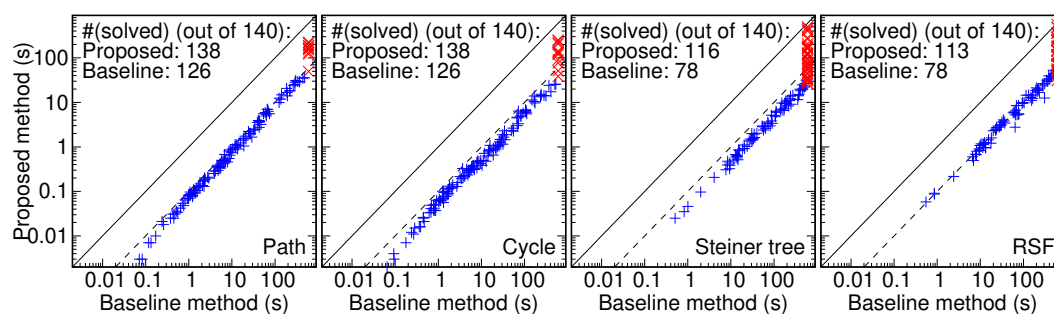
7 Experiments

We empirically compared the proposed and the baseline methods with respect to the computational time. Here the baseline method is to separately build a ZDD by FBS for each constraint. Both methods were implemented in C++ and compiled by g++ with -O3 option. We used TdZdd [14] for the baseline method, which is a highly optimized C++ library for FBS. We also used TdZdd for the proposed method to construct ZDD $Z_{\mathcal{F}}$ of base set. Experiments are conducted on a single thread of a Linux machine with AMD EPYC 7763 2.45 GHz CPU and 2048 GB RAM; note that we used less than 256 GB of memory during the experiments. We set the time limit of every run to 600 seconds.

We used both synthetic graphs and real benchmarks as tested graphs. The synthetic ones are grid graphs; Grid- $w \times h$ represents a grid graph with $w \times h$ vertices. For the others, we used the Rocketfuel [29] and Romegraph datasets [7]. Rocketfuel was also used in [22]. From Romegraph, we chose all the graphs with $n = 100$: there were 140 such graphs. Identical edge ordering was used for both methods, and it was decided as follows: For the grid graphs, we used the edge ordering of Iwashita et al. [16], which is better for the DP on grid graphs. For the other graphs, we used beam-search heuristics [13] to determine the edge ordering.

We evaluated four problem settings in Section 2: path, cycle, Steiner tree, and rooted spanning forest (RSF). The given vertices for these settings were determined as follows. Let $d(v, v')$ be the shortest distance between vertices v and v' . For the path problem, we chose the most distant vertex pair as s, t , i.e., s, t satisfies $d(s, t) = \max_{v, v'} d(v, v')$. For the cycle problem, we chose the graph center as s , i.e., $s \in \text{argmin}_v \max_{v'} d(v, v')$. For the other problems, we chose four vertices as T such that the sum of the distances between distinct vertices, $\sum_{v, v' \in T: v \neq v'} d(v, v')$, is maximized.

Table 2 shows the result for the grid graphs and the Rocketfuel dataset. For all the graphs and problem settings solved by both methods within the time limit, the proposed method ran about 10–20 times faster than the baseline method. The complexity analyses in Section 6



■ **Figure 3** Computational time for Romegraph dataset: Blue points indicate instances solved by both methods, and red points indicate those solved only by proposed method. Solid black lines indicate elapsed time for both methods is identical, and dashed lines indicate proposed method is 10 times faster than baseline method.

suggest that the proposed method becomes faster than the existing method when n is large. Table 2 exhibits such a tendency. For example, for the Grid- $8 \times h$ graphs, the proposed method becomes much faster than the baseline method when $n = 8h$ is increased. In addition, both methods ran faster for graphs with smaller fw value, reflecting the complexity analyses.

Fig. 3 plots the result for the Romegraph dataset and also describes the number of graphs solved by each method within the time limit. Here each point corresponds to a graph, where the blue ones are those solved by both methods and the red ones are those solved only by the proposed method. Although the computational time itself varied from less than 0.01 to 600 seconds, for almost all the graphs the proposed method ran about 10–20 times faster than the existing method. This ratio is kept because the graphs all have the same number of vertices: 100. We give detailed results for Romegraph in Appendix A.4.

8 Conclusion

We proposed a novel framework, compDP, for solving multiple subgraph counting problems with similar connectivity constraints simultaneously. A complexity analysis showed that the proposed method ran $O(n)$ times faster than the baseline approach, and the experiments revealed the proposed method’s efficiency.

As a future work, we will consider dealing with the reachability in directed graphs. There are approaches for building BDDs of reachability constraints [19, 30], and we want to consider whether they can be incorporated into our framework.

References

- 1 N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.
- 2 Eric T. Bax. Algorithms to count paths and cycles. *Information Processing Letters*, 52(5):249–252, 1994.
- 3 Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, C-35(8):677–691, 1986.
- 4 Radu Curticapean. Counting problems in parameterized complexity. In *Proc. of the 13th International Symposium on Parameterized and Exact Computation (IPEC)*, pages 1:1–1:18, 2018.
- 5 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

- 6 Luigi Fratta and Ugo Montanari. A Boolean algebra method for computing the terminal reliability in a communication network. *IEEE Trans. Circuit Theory*, 20(3):203–211, 1973.
- 7 graphdrawing.org. RomeGraph. <http://www.graphdrawing.org/data.html>.
- 8 G. Hardy, C. Lucet, and N. Limnios. K-terminal network reliability measures with binary decision diagrams. *IEEE Trans. Rel.*, 56(3):506–515, 2007.
- 9 J.U. Herrmann. Improving reliability calculation with augmented binary decision diagrams. In *Proc. of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 328–333, 2010.
- 10 Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.
- 11 Takeru Inoue. Reliability analysis for disjoint paths. *IEEE Trans. Rel.*, 68(3):985–998, 2018.
- 12 Takeru Inoue, Norihito Yasuda, Shunsuke Kawano, Yuji Takenobu, Shin-ichi Minato, and Yasuhiro Hayashi. Distribution network verification for secure restoration by enumerating all critical failures. *IEEE Trans. Smart Grid*, 6(2):843–852, 2015.
- 13 Yuma Inoue and Shin-ichi Minato. Acceleration of ZDD construction for subgraph enumeration via pathwidth optimization. Technical Report TCS-TR-A-16-80, Division of Computer Science, Hokkaido University, 2016.
- 14 Hiroaki Iwashita, Jun Kawahara, and Kohei Shinohara. TdZdd. <https://github.com/kunisura/TdZdd>.
- 15 Hiroaki Iwashita and Shin-ichi Minato. Efficient top-down ZDD construction techniques using recursive specifications. Technical Report TCS-TR-A-13-69, Division of Computer Science, Hokkaido University, 2013.
- 16 Hiroaki Iwashita, Yoshio Nakazawa, Jun Kawahara, Takeaki Uno, and Shin-ichi Minato. Efficient computation of the number of paths in a grid graph with minimal perfect hash functions. Technical Report TCS-TR-A-13-64, Division of Computer Science, Hokkaido University, 2013.
- 17 Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Trans. Fundamentals*, E100-A(9):1773–1784, 2017.
- 18 Donald E. Knuth. *The art of computer programming: Vol. 4A. Combinatorial Algorithms, Part 1*. Addison-Wesley Professional, 2011.
- 19 Takanori Maehara, Hirofumi Suzuki, and Masakazu Ishihata. Exact computation of influence spread by binary decision diagrams. In *Proc. of the 26th International World Wide Web Conference (WWW)*, pages 947–956, 2017.
- 20 Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th ACM/IEEE Design Automation Conference (DAC)*, pages 272–277, 1993.
- 21 Fred Moskowitz. The analysis of redundancy networks. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 77(5):627–632, 1958.
- 22 Kengo Nakamura, Takeru Inoue, Masaaki Nishino, and Norihito Yasuda. Efficient network reliability evaluation for client-server model. In *Proc. of IEEE Global Communication Conference (GLOBECOM)*, pages 1–6, 2021.
- 23 R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.
- 24 Neil Robertson and P.D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.
- 25 Arnie Rosenthal. Computing the reliability of complex networks. *SIAM J. Appl. Math.*, 32(2):384–393, 1977.
- 26 Shinsaku Sakaue and Kengo Nakamura. Differentiable equilibrium computation with decision diagrams for Stackelberg models of combinatorial congestion games. In *Proc. of the 35th Conference on Neural Information Processing Systems (NeurIPS)*, pages 9416–9428, 2021.

- 27 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.
- 28 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.
- 29 Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, 2004.
- 30 Hirofumi Suzuki, Masakazu Ishihata, and Shin-ichi Minato. Exact computation of strongly connected reliability by binary decision diagrams. In *Proc. of the 12th Annual International Conference on Combinatorial Optimization and Applications (COCOA)*, pages 281–295, 2018.
- 31 S. Tsukiyama, I. Shirakawa, H. Ozaki, and H. Ariyoshi. An algorithm to enumerate all cutsets of a graph in linear time per cutset. *J. ACM*, 27(4):619–632, 1980.
- 32 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.
- 33 R. Kevin Wood. Factoring algorithms for computing K-network network reliability. *IEEE Trans. Rel.*, 35(3):269–278, 1986.

A Appendix

A.1 Treatment of Degree 1 Vertices

As in Section 5.1, we can compute the $\text{count}(v)$ value by focusing on the i -th level of Z where $v \in F_i$. However, if v 's degree is 1, no such i exists. Let $e_i = \{v, v'\}$ be the only edge incident to v . Then for $i' < i$, v is not present in $E_{>i'}$, and for $i' \geq i$, it is not present in $E_{<i'}$, so no frontier contains v . Therefore, we have an alternative formula for computing $\text{count}(v)$ like Eq. (6). Let us focus on the i -th level where $e_i = \{v, v'\}$ is the only edge incident to v .

First, we address the case where the other endpoint, v' , is in F_i . Let $\hat{n} \in L_i$ be an arbitrarily chosen i -th level node of Z . Since e_i is the only edge incident to v , if e_i is excluded, v remains as an isolated vertex. Thus, if $V_{P^*}' \neq \emptyset$, constraint $(\#_v)$ will never be satisfied. Otherwise, if $V_{P^*}' = \emptyset$, constraint $(\#_v)$ is always satisfied. In this case, the set of paths in $\mathcal{R}_Z(\hat{r}, \top)$ that passes \hat{n} whose corresponding subgraph satisfies $(\#_v)$, i.e., $\mathcal{R}_{v, \hat{n}}$, can be written as $\mathcal{R}_Z(\hat{r}, \hat{n}) \sqcup \{\text{lo}(\hat{n})\} \sqcup \mathcal{R}_Z(\hat{n}^-, \top)$. The sum of their path products is $\hat{n} \cdot \mathbf{p} \cdot w_{e_i}^- \cdot \hat{n}^- \cdot \mathbf{r}$ given that $\hat{n}^- \neq \perp$. If e_i is included, v is connected with v' , and so condition $(\#_v)$ is met if and only if condition $(\#'_B)$ for $B = \hat{n} \cdot \text{comp}[v']$ is met. In this case, the set of paths in $\mathcal{R}_Z(\hat{r}, \top)$ that passes \hat{n} whose corresponding subgraph satisfies $(\#_v)$ can be written as $\mathcal{R}_Z(\hat{r}, \hat{n}) \sqcup \mathcal{R}_{\hat{n}, \hat{n} \cdot \text{comp}[v']}'$, where $\mathcal{R}_{\hat{n}, B}'$ is the set of paths in $\mathcal{R}_{\hat{n}, B}$ that passes through $\text{hi}(\hat{n})$. The sum of their path products is $\hat{n} \cdot \mathbf{p} \cdot \hat{n} \cdot \mathbf{q}^+[\hat{n} \cdot \text{comp}[v']]$ using the notion \mathbf{q}^+ introduced in Section 5.2. The value $\text{count}(v)$ can be computed by their sum over $\hat{n} \in L_i$:

$$\text{count}(v) = \sum_{\hat{n} \in L_i} \hat{n} \cdot \mathbf{p} \cdot \hat{n} \cdot \mathbf{q}^+[\hat{n} \cdot \text{comp}[v']] + \begin{cases} 0 & (V_{P^*}' \neq \emptyset) \\ \sum_{\hat{n} \in L_i: \hat{n}^- \neq \perp} \hat{n} \cdot \mathbf{p} \cdot w_{e_i}^- \cdot \hat{n}^- \cdot \mathbf{r} & (V_{P^*}' = \emptyset) \end{cases}. \quad (10)$$

The remaining issue is how to cope with case $v' \notin F_i$. For it, we can assume $v' \in F_{i+1}$; otherwise, v' is also a degree 1 vertex that means graph G consists of only e_i since G is connected, which is trivial. The case where e_i is excluded is treated in the same way as above. If e_i is included, let \hat{n} be an arbitrary i -th level node of Z . Since v is connected with v' , constraint $(\#_v)$ is met if and only if constraint $(\#'_B)$ for $B = \hat{n}^- \cdot \text{comp}[v']$ is met. Therefore, the set of paths in $\mathcal{R}_Z(\hat{r}, \top)$ that passes \hat{n} whose corresponding subgraph satisfies $(\#_v)$ can be written as $\mathcal{R}_Z(\hat{r}, \hat{n}) \sqcup \{\text{hi}(\hat{n})\} \sqcup \mathcal{R}_{\hat{n}^+, \hat{n}^+ \cdot \text{comp}[v']}'$, given that $\hat{n}^+ \neq \perp$. The sum of their path products is $\hat{n} \cdot \mathbf{p} \cdot w_{e_i}^+ \cdot \hat{n}^+ \cdot \mathbf{q}^+[\hat{n}^+ \cdot \text{comp}[v']]$. The value $\text{count}(v)$ can be computed by

■ **Algorithm 3** Frontier-based search with subsetting for connectivity constraint P and base ZDD $Z_{\mathcal{F}}$ representing the base family of subgraphs \mathcal{F} .

```

1  $\hat{r}.\text{comp} \leftarrow \{\}, \hat{r}.\text{vset} \leftarrow P, \hat{r}.\text{base} \leftarrow \hat{r}_{\mathcal{F}}$  ( $Z_{\mathcal{F}}$ 's root),  $L_1 \leftarrow \{\hat{r}\}, L_i \leftarrow \emptyset$  ( $i = 2, \dots, m+1$ )
2 for  $i \leftarrow 1$  to  $m$  do //  $e_i = \{v, v'\}$ 
3   foreach  $\hat{n} \in L_i$  do
4     foreach  $f \in \{-, +\}$  do
5        $\hat{n}' \leftarrow \hat{n}$ 
6       if  $f = +$  and  $i < \text{lb}(\hat{n}.\text{base})$  then
7          $\hat{n}' \leftarrow \perp$  and goto finish // No further subgraphs in  $\mathcal{F}$ 
8       if  $i = \text{lb}(\hat{n}.\text{base})$  then
9          $\hat{n}.\text{base} \leftarrow (\hat{n}.\text{base})^f$  // base proceeds to child node
10        if  $\hat{n}.\text{base} = \perp$  then
11           $\hat{n}' \leftarrow \perp$  and goto finish // No further subgraphs in  $\mathcal{F}$ 
12        if  $f = +$  and  $v, v' \in V_{\hat{n}.\text{vset}}$  and  $\hat{n}.\text{vset}[v] \neq \hat{n}.\text{vset}[v']$  then
13           $\hat{n}' \leftarrow \perp$  and goto finish //  $v$  and  $v'$  must not be connected
14        foreach  $u \in \{v, v'\} \setminus F_i$  do // Vertices entering the frontier
15           $\hat{n}.\text{comp} \leftarrow \hat{n}.\text{comp} \cup \{\{u\}\}$  // Add  $u$  as an isolated vertex
16        if  $f = +$  and  $\hat{n}.\text{comp}[v] \neq \hat{n}.\text{comp}[v']$  then // Connecting two components
17          Merge  $\hat{n}.\text{comp}[v]$  and  $\hat{n}.\text{comp}[v']$  into one
18          if  $v \in V_{\hat{n}.\text{vset}}$  and  $v' \notin V_{\hat{n}.\text{vset}}$  then Add vertices in  $\hat{n}.\text{comp}[v']$  to  $\hat{n}.\text{vset}[v]$ 
19          if  $v \notin V_{\hat{n}.\text{vset}}$  and  $v' \in V_{\hat{n}.\text{vset}}$  then Add vertices in  $\hat{n}.\text{comp}[v]$  to  $\hat{n}.\text{vset}[v']$ 
20        foreach  $u \in \{v, v'\} \setminus F_{i+1}$  do // Vertices leaving the frontier
21          if  $\{u\} \in \hat{n}.\text{comp}$  then // Component containing  $u$  leaves frontier
22            if  $u \in V_{\hat{n}.\text{vset}}$  and  $\{u\} \notin \hat{n}.\text{vset}$  and  $\{u, *\} \notin \hat{n}.\text{vset}$  then
23               $\hat{n}' \leftarrow \perp$  and goto finish //  $u$  must be connected to  $w \in \hat{n}.\text{vset}[u]$ 
24            Remove  $u$  from  $\hat{n}.\text{comp}$  and  $\hat{n}.\text{vset}$  if exists
25        if  $\hat{n}' \neq \perp$  then
26          if  $i = m$  then  $\hat{n}' \leftarrow \top$  // All conditions are meet
27          else if  $\exists \hat{n}'' \in L_{i+1}$  s.t.  $\hat{n}''.\text{comp} = \hat{n}.\text{comp}$  and  $\hat{n}''.\text{vset} = \hat{n}.\text{vset}$  and
                 $\hat{n}''.\text{base} = \hat{n}.\text{base}$  then
28             $\hat{n}' \leftarrow \hat{n}''$  // Already generated node
29          else  $L_{i+1} \leftarrow L_{i+1} \cup \{\hat{n}'\}$  // Newly generated node
30        finish:
31         $\hat{n}^f \leftarrow \hat{n}'$  // Set lo- or hi-child of  $\hat{n}$  to  $\hat{n}'$ 

```

$$\text{count}(v) = \sum_{\hat{n} \in L_i: \hat{n}^+ \neq \perp} \hat{n}.\text{p} \cdot w_{e_i}^+ \cdot \hat{n}^+ \cdot \text{q}[\hat{n}^+.\text{comp}[v']] + \begin{cases} 0 & (V_{P^*} \neq \emptyset) \\ \sum_{\hat{n} \in L_i: \hat{n}^- \neq \perp} \hat{n}.\text{p} \cdot w_{e_i}^- \cdot \hat{n}^- \cdot r & (V_{P^*} = \emptyset) \end{cases}. \quad (11)$$

A.2 Pseudocode for FBS with Subsetting

Next we explain the subsetting [15], which is used for taking the set intersection of the ZDDs, and describe the pseudocode for the FBS with subsetting. Given ZDD $Z_{\mathcal{F}}$ that represents the base set \mathcal{F} and connectivity constraint P , it constructs a ZDD that represents $\mathcal{F} \cap \mathcal{C}(P)$. Starting with $\hat{r}_{\mathcal{F}}$ where $\hat{r}_{\mathcal{F}}$ is the root node of $Z_{\mathcal{F}}$, we traverse the lo-arc of $Z_{\mathcal{F}}$ if e_i is excluded in the FBS and its hi-arc of if e_i is included in the FBS. We now record the present node in $Z_{\mathcal{F}}$ as $\hat{n}.\text{base}$ for each node \hat{n} . If it reaches \perp in $Z_{\mathcal{F}}$, there are no further subgraphs in \mathcal{F} , and so pruning is executed. Here we can identify the two nodes of Z if their base as well as comp and vset are identical.

Based on these procedures, the FBS with subsetting can be described as Algorithm 3. Here the red part is newly added elements that are not included in Algorithm 1. Note that these codes are a bit complicated than the above explanation since we also cope with the case where $Z_{\mathcal{F}}$ is not normalized.

A.3 Proof of Propositions in Complexity Analysis

Proof of Proposition 2. By storing `comp` and `vset` as an integer sequence whose length is `fw`, as in the previous work [22], FBS with an intersection runs in $O(\text{fw} \cdot |Z|)$ time where Z is the resultant ZDD. Since $|Z|$ can be bounded by the product of $|Z_{\mathcal{F}}|$ and $|Z_{\text{FBS}(P)}|$ [20], $|Z| = O(|Z_{\mathcal{F}}| |Z_{\text{FBS}(P^*)}|)$ in the proposed algorithm. For each node, `p`, `r`, and `q[B]` can be computed in constant time, and there are at most `fw` sets in `comp`. Thus, the DP computation is completed in $O(\text{fw} \cdot |Z|)$ time. The complete computation of $\text{count}(v)$ can be done in $O(|Z|)$ time; by choosing i such that e_i is the first edge containing v for every v , each level of Z is scanned at most twice for computing $\text{count}(v)$ for every v . ◀

Proof of Proposition 3. We consider the number of possible patterns for the `comp` and `vset` pair. We focus on an i -th level. Since `comp` is simply a partition of F_i , the number of possible patterns for it is $D_{|F_i|}$. The number of possible patterns for `vset` can be bounded in two ways. First, `vset` retains the information of how the components in `vset` are connected to each set in P . Each component of `comp` is connected to at most one set in P , since if more than two sets are connected, connectivity constraint P is violated. Since there are at most $|F_i|$ components, the number of `vset` patterns is bounded by $(c_P + 1)^{|F_i|}$, where $+1$ deals with the case where no component is connected to any sets in P . Second, `vset` can be seen as retaining the information of how the vertices in P are connected to the component in `comp`. Thus, the number of `vset` patterns is bounded by $(|F_i| + 1)^{v_P}$, where $+1$ deals with the case where a vertex in P is not connected to any component in `comp`. To sum up, the number of patterns of the `comp` and `vset` pair can be bounded by $O(D_{|F_i|} \cdot \min\{(c_P + 1)^{|F_i|}, (|F_i| + 1)^{v_P}\})$.

Since there are m levels and $|F_i| \leq \text{fw}$, the overall size is bounded by $O(m D_{\text{fw}} \cdot \min\{(c_P + 1)^{\text{fw}}, (\text{fw} + 1)^{v_P}\})$. ◀

Proof of Proposition 4. Since running Algorithm 1 with P^* and $P^* \sqcup$ yields the same representing family of sets, $\mathcal{C}(P^* \sqcup)$, we only have to address the number of patterns of `comp` and `vset`. The only difference is that when running FBS with P^* , we must determine which set in `vset` has $*$. Since there are at most c_{P^*} sets in `vset`, there will be at most c_{P^*} patterns of `vset` for a node in $Z_{\text{FBS}(P^* \sqcup)}$ when running FBS with P^* . Thus, $|Z_{\text{FBS}(P^*)}| \leq c_{P^*} |Z_{\text{FBS}(P^* \sqcup)}|$ holds. ◀

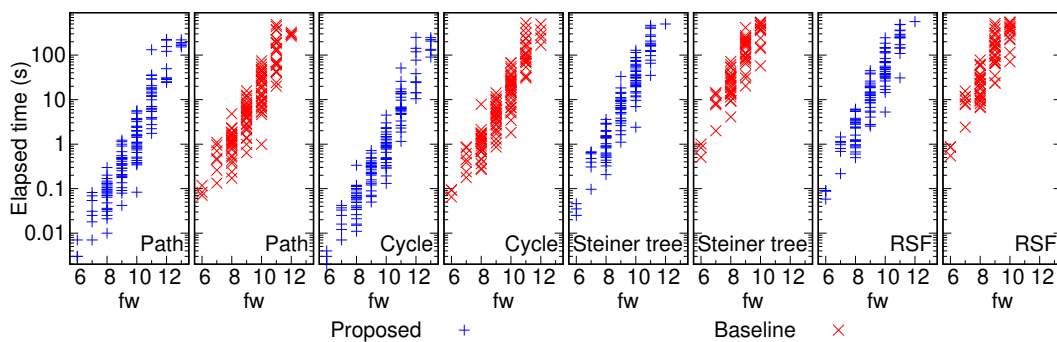
A.4 Detailed Experimental Results for Romegraph Dataset

Next we describe a detailed experimental results for the Romegraph dataset, which has 140 graphs whose number of nodes is exactly 100. With beam-search heuristics [13], the frontier width `fw` of each graph ranges from 6 to 14. The number of graphs per value of `fw` is described in Table 3.

Table 3 also shows the number of graphs solved within the time limit for each method, each problem setting, and each frontier width value. In addition, Fig. 4 plots the computational time for the Romegraph dataset aggregated by frontier width `fw`. For both methods, the graphs with a larger `fw` value are clearly difficult to solve, i.e., time-consuming; this outcome reflects the complexity results in Section 6. However, our proposed method can also treat graphs with a larger `fw` value than the baseline method. This again clearly indicates the efficiency of our proposed method.

■ **Table 3** Comparison of the number of solved graphs in Romegraph dataset within time limit for each frontier width.

fw	#(graphs)	Path		Cycle		Steiner tree		RSF	
		Ours	Base	Ours	Base	Ours	Base	Ours	Base
6	3	3	3	3	3	3	3	3	3
7	7	7	7	7	7	7	7	7	7
8	26	26	26	26	26	26	26	26	26
9	28	28	28	28	28	28	27	28	28
10	33	33	33	33	33	33	15	33	14
11	25	25	24	25	24	18	0	15	0
12	10	10	5	10	5	1	0	1	0
13	6	6	0	6	0	0	0	0	0
14	2	0	0	0	0	0	0	0	0
Total	140	138	126	138	126	116	78	113	78



■ **Figure 4** Computational time for Romegraph dataset aggregated by frontier width fw. Blue points indicate results of proposed method, and red points indicate results of baseline method.