Linear-Delay Enumeration for Minimal Steiner Problems

Yasuaki Kobayashi
Hokkaido University
Hokkaido, Japan
koba@ist.hokudai.ac.jp

Kazuhiro Kurita
National Institute of Informatics
Tokyo, Japan
kurita@nii.ac.jp

Kunihiro Wasa
Toyohashi University of Technology
Aichi, Japan
wasa@cs.tut.ac.jp

ABSTRACT

Kimelfeld and Sagiv [Kimelfeld and Sagiv, PODS 2006], [Kimelfeld and Sagiv, Inf. Syst. 2008] pointed out that the problem of enumerating K-fragments is of great importance in a keyword search on data graphs. In a graph-theoretic term, the problem corresponds to enumerating minimal Steiner trees in (directed) graphs. In this paper, we propose a linear-delay and polynomial-space algorithm for enumerating all minimal Steiner trees, improving on a previous result in [Kimelfeld and Sagiv, Inf. Syst. 2008]. Our enumeration algorithm can be extended to other Steiner problems, such as minimal Steiner forests, minimal terminal Steiner trees, and minimal directed Steiner trees. As another variant of the minimal Steiner tree enumeration problem, we study the problem of enumerating minimal induced Steiner subgraphs. We propose a polynomial-delay and exponential-space enumeration algorithm of minimal induced Steiner subgraphs on claw-free graphs. Contrary to these tractable results, we show that the problem of enumerating minimal group Steiner trees is at least as hard as the minimal transversal enumeration problem on hypergraphs.

CCS CONCEPTS

• **Mathematics of computing → Graph enumeration**.

KEYWORDS

Enumeration, K-fragment, Polynomial-delay, Steiner tree

1 INTRODUCTION

Kimelfeld and Sagiv [26] observed that enumerating K-fragments for a set of keywords K in data graphs is a core component in several keyword search systems. A data graph is a graph that consists of two types of nodes: structural nodes and keyword nodes, and each keyword node corresponds to some keyword in K. A K-fragment is a subtree in a data graph that contains all keyword nodes for K and no proper subtree that contains them. There are several types of K-fragments, *undirected K-fragments*, *strong K-fragments*, and *directed K-fragments*. In a graph-theoretic term, they are equivalent to *Steiner trees*, *terminal Steiner trees*, and *directed Steiner trees*, respectively.

Given an undirected graph $G = (V, E)$ and a subset of vertices $W \subseteq V$, called *terminals*, the *Steiner tree problem* asks to find a Steiner tree of (G, W) that has a minimum number of edges. Here, a *Steiner tree* of (G, W) is a subtree of G that contains all terminals. The Steiner tree problem is a classical combinatorial optimization problem and has arisen in several areas [4, 15, 25, 26, 28]. This problem is shown to be NP-hard in Karp's seminal work [22] and has been studied from several perspectives, such as approximation algorithms [5, 14], parameterized algorithms [11], and algorithms in practice [2, 20, 30]. There are also many variants of this problem. See [17], for a compendium on variants of the Steiner tree problem.

The notion of Steiner trees emerges in investigating the connectivity or reachability of a specified vertex subset in networks. Steiner trees can be seen as a generalization of some basic combinatorial structures in graphs, such as s-t (shortest) paths and spanning trees. In some applications, it is preferable to find *multiple* solutions rather than a single solution. E.g., the problem of finding k distinct shortest s-t paths has been widely studied [12, 18, 34] due to various practical applications.

However, as mentioned before, finding a *minimum* Steiner subgraph is intractable in general. Motivated by these facts, we focus on enumerating *minimal* Steiner subgraphs. We say that a Steiner subgraph H of (G, W) is *minimal* if there is no proper subgraph of H that is a Steiner subgraph of (G, W). It is easy to see that every minimal Steiner subgraph of (G, W) forms a tree, but some Steiner trees of (G, W) may not be minimal Steiner subgraphs of (G, W). In this paper, we address the problem of enumerating minimal Steiner trees of (G, W), which is defined as follows.

DEFINITION 1 (STEINER TREE ENUMERATION). *Given an undirected graph $G = (V, E)$ and a terminal set $W \subseteq V$, the task is to enumerate all the minimal Steiner trees of (G, W).*

There are several known results on enumeration algorithms related to STEINER TREE ENUMERATION and its variants [8,10,24–26]. See Table 1 for details. We say that a Steiner tree T of (G, W) is a *terminal Steiner tree* of (G, W) if every terminal in W is a leaf of T. For a collection of terminal sets $\mathcal{W} = \{W_1, \ldots, W_s\}$, where $W_i \subseteq V$, we say that a subgraph of G is a *Steiner forest* of $(G, \mathcal{W})$ if for each $1 \leq i \leq s$, the subgraph has a component that is a Steiner subgraph of (G, W_i). For a directed graph $D = (V, E)$, a terminal set W, and $r \in V \setminus W$, we say that a subgraph of D is a *directed Steiner tree* of (D, W, r) if the subgraph contains directed path from r to each $w \in W$. Since minimal subgraphs of each variant form trees, forests, and directed trees, these are called terminal Steiner trees, Steiner forests, and directed Steiner trees, respectively.

For enumeration problems, *enumeration algorithms* are required to generate all the solutions one by one without duplication. Since enumeration problems generally have an exponential number of solutions, we measure the complexity of enumeration algorithms

Enumeration problem	Time	Preprocessing	Space
STEINER TREE [26]	$O(m(|T_i| + |T_{i-1}|))^{\dagger}$	$O(m|T_1|)$	$O(nm)$
STEINER FOREST [24]	inc. poly.	poly.	exp.
TERMINAL STEINER TREE [26]	$O(m(|T_i| + |T_{i-1}|))^{\dagger}$	$O(m|T_1|)$	$O(nm)$
DIRECTED STEINER TREE [26]	$O(mt(|T_i| + |T_{i-1}|))^{\dagger}$	$O(mt|T_1|)$	$O(nm)$
INDUCED STEINER SUBGRAPH with $t \leq 3$ [8]	$O(mn^3)$ amortized	$O(mn^3)$	poly.
MINIMUM STEINER TREE [10]	$O(n)$ delay	$O(n^{t-2} + n^2m)$	$O(n^{t-2} + n^2m)$
MINIMUM INDUCED STEINER SUBGRAPH [8]	$O\left(m2^{3t^2}\right)$ amortized	$O(n^{t-2} + n^2m)$	$O(n^{t-2} + n^2m)$
STEINER TREE [This work]	$O(n + m)$ delay	$O(n(n+m))$	$O(n^2)$
STEINER FOREST [This work]	$O(n + m)$ delay	$O(n(n+m))$	$O(n^2)$
TERMINAL STEINER TREE [This work]	$O(n + m)$ delay	$O(n(n+m))$	$O(n^2)$
DIRECTED STEINER TREE [This work]	$O(n + m)$ delay	$O(n(n+m))$	$O(n^2)$
INDUCED STEINER SUBGRAPH on claw-free graphs [This work]	poly. delay	poly.	exp.

Table 1: The table summarizes known and our results for STEINER TREE ENUMERATION and related problems. Let n, m, and t be the number of vertices, edges, and terminals, respectively. $|T_i|$ the number of edges in the i-th solution. Note that the algorithms for MINIMUM STEINER TREE and MINIMUM INDUCED STEINER SUBGRAPH enumerate all *minimum* Steiner trees, whereas algorithms for other problems enumerate *minimal* Steiner trees. The running time marked † indicates the delay between the $(i-1)$-th and the i-th solutions.

in terms of the input size n and the output size N (i.e., the number of solutions). The *delay* between two consecutive solutions of an enumeration algorithm is the running time interval between them. The delay of an enumeration algorithm is the worst delay between every pair of consecutive solutions. Note that, unless stated otherwise, the running time before generating the first solution and after generating the last solution is upper bounded by the delay. If the delay between $(i-1)$-th solution and i-th solution is bounded by $O(poly(n+i))$, then we call such an enumeration algorithm an *incremental-polynomial time algorithm*. An enumeration algorithm is called a *polynomial-delay enumeration algorithm* if the delay is upper bounded by a polynomial in n. Finally, an enumeration algorithm that runs in total $O(N \cdot poly(n))$ time is called an *amortized polynomial-time algorithm*.

Kimelfeld and Sagiv developed three efficient enumeration algorithms for STEINER TREE ENUMERATION and its variants [26]. Their algorithms for enumerating minimal Steiner trees, minimal terminal Steiner trees, and minimal directed Steiner trees run in $O(m(|T_i| + |T_{i-1}|))$, $O(m(|T_i| + |T_{i-1}|))$, and $O(mt(|T_i| + |T_{i-1}|))$ delay between $(i-1)$-th and i-th solutions, respectively. Here, m is the number of edges in the input graph, $|T_i|$ is the number of edges in the i-th solution, and t is the number of terminals, respectively. They also proposed efficient algorithms for enumerating Steiner trees and their variants in an "approximate" ascending order of their weights [25]. Khachiyan et al. [24] studied the problem of enumerating minimal Steiner forests in graphs as a special case of the circuit enumeration problem on matroids and gave an incremental-polynomial time enumeration algorithm for this problem.

Our contribution: Here, we give our main result for STEINER TREE ENUMERATION.

THEOREM 2. *There is an $O(n + m)$ delay and $O(n^2)$ space[1] enumeration algorithm for STEINER TREE ENUMERATION provided that*

we are allowed to use $O(n(n+m))$ preprocessing time, where n and m are the number of vertices and edges, respectively. Moreover, without additional preprocessing time, we can implement the algorithm so that it runs in time $O(n + m)$ amortized time per solution and $O(n + m)$ space.

This algorithm can be extended to enumerating minimal Steiner forests, minimal terminal Steiner trees, and minimal directed Steiner trees. Our algorithms for these problems achieve the same running time bounds as in Theorem 2. (See Theorems 25, 31 and 36.)

The basic idea of these algorithms is a standard branching technique. Starting from an arbitrary terminal, we recursively grow a partial Steiner tree T by attaching a path between T and a terminal not contained in T. Since the paths between T and a terminal w can be enumerated in polynomial delay, we immediately have a polynomial-delay and polynomial-space algorithm for STEINER TREE ENUMERATION. To improve the running time and space complexity of this simple algorithm, we carefully design the entire branching strategy and the path enumeration algorithm. To this end, we employ Read and Tarjan's path enumeration algorithm [29] and discuss the delay and space complexity of their algorithm, which is not discussed explicitly in [29]. As for the delay complexity, we can easily obtain a linear delay bound by applying a standard technique [26, 33]. However, to obtain a linear space bound for STEINER TREE ENUMERATION, we need a nontrivial modification of their algorithm, which will be discussed in Section 3. We also exploit the output queue method due to Uno [33] to improve the delay for STEINER TREE ENUMERATION.

In contrast to these tractable cases, we also show that two variants of STEINER TREE ENUMERATION are "not so easy": the problem of enumerating internal Steiner trees is NP-hard, and that of enumerating group Steiner trees is at least as hard as the minimal hypergraph transversal problem (see Sections 2 and 6 for their definitions). It should be mentioned that the polynomial-time solvability for the minimal hypergraph transversal enumeration problem with respect to the input size n and output size N (i.e., $(n+N)^{O(1)}$) is one

---

[1]Whenever we refer to space complexity, we assume the RAM model, that is, a single word of memory contains $O(\log n)$ bits.

of the most challenging open problems in the field of enumeration algorithms and the best known algorithm runs in quasi-polynomial time $(n + N)^{o(\log(n+N))}$ [13].

We also consider the problem of enumerating minimal induced Steiner subgraphs as another variant of STEINER TREE ENUMERATION. This problem is known to be at least as hard as the minimal transversal enumeration problem on hypergraphs even when the input is restricted to split graphs [8]. They also showed that if the number of terminals is at most three, one can solve this problem in $O(mn^3)$ amortized time per solution and polynomial space. In this paper, we develop a polynomial-delay and exponential-space enumeration algorithm on claw-free graphs with an arbitrary number of terminals. Since the class of claw-free graphs is a superclass of line graphs [1] and STEINER TREE ENUMERATION is a special case of the problem of enumerating minimal induced Steiner subgraphs on line graphs, our result non-trivially expands the tractability of Steiner subgraph enumeration.

2 PRELIMINARIES

Let $G = (V, E)$ be an undirected (or directed) graph with vertex set V and edge set E. We also denote by $V(G)$ and $E(G)$ the sets of vertices and edges in G, respectively. Throughout the paper, we assume that graphs have no self-loops but may have multiedges. For a vertex v, we denote by $N_G(v)$ the set of neighbors of v in G and by $\Gamma_G(v)$ the set of incident edges of v. These notations are extended to sets: For $U \subseteq V$, $N_G(U) = \bigcup_{v \in U} N_G(v) \setminus U$ and $\Gamma_G(U) = \bigcup_{v \in U} \Gamma_G(v) \setminus \{\{u, v\} : u, v \in U\}$. If there is no confusion, we drop the subscript G from these notations. For an edge $e = \{u, v\}$, the graph obtained from G by contracting e is denoted by G/e, that is, $G/e = (V', E')$, where $V' = (V \setminus \{u, v\}) \cup \{w\}$ and $E' = (E \setminus (\Gamma_G(u) \cup \Gamma_G(v))) \cup \{\{x, w\} : \{x, y\} \in \Gamma_G(u) \cup \Gamma_G(v) \wedge (y = u \vee y = v)\}$. In this notation, the set E' is considered as a multiset and hence G/e may have multiedges. For a set of edges $F \subseteq E$, the (multi)graph G/F is obtained from G by contracting all edges in F.

For $U \subseteq V$, $G[U]$ is the subgraph of G induced by U, that is, $G[U] = (U, E')$ with $E' = \{\{u, v\} \in E : u, v \in U\}$. For $F \subseteq E$, we also use $G[F]$ to denote the subgraph of the form $(V(F), F)$, where $V(F)$ is the set of end vertices of edges in F. Let H be a subgraph of G. For $F \subseteq E(G) \setminus E(H)$, we denote by $H + F$ the graph $G[E(H) \cup F]$. Similarly, for $F \subseteq E(H)$, we denote a subgraph $G[E(H) \setminus F]$ as $H - F$. When F consists of a single edge e, we simply write $H + e$ and $H - e$ to denote $H + F$ and $H - F$, respectively. In this paper, we may identify a subgraph with its edges.

Let X and Y be disjoint sets of vertices of G. A path P is called an X-Y path if an end vertex of P belongs to X, the other end belongs to Y, and every internal vertex of P does not belong to $X \cup Y$. If X and/or Y are singletons u and v, respectively, we may write u-Y, X-v, u-v paths to denote an X-Y path.

Let $W \subseteq V$ be a set of terminals. A subgraph H of G is a *Steiner subgraph* of (G, W) if there is a path between every pair of vertices of W in H. Note that every minimal Steiner subgraph forms a tree, called a *minimal Steiner tree*. The following characterization is straightforward from the observation that every leaf of a minimal Steiner tree belongs to W as otherwise, we can delete it.

PROPOSITION 3. *A Steiner tree T of (G, W) is minimal if and only if every leaf of T is a terminal.*

Let $\mathcal{W} = \{W_i \subseteq V : 1 \le i \le s\}$ be a family of terminal sets. A subgraph of G is called a *Steiner forest* of $(G, \mathcal{W})$ if it has no cycles and there is a path between every pair of vertices in W_i for $1 \le i \le s$ in the forest. We say that a Steiner forest is *minimal* if it has no any Steiner forest as a subgraph. A subgraph of G is called a *group Steiner tree* of $(G, \mathcal{W})$ if it is a tree and there is a path between *some* $u \in W_i$ and *some* $v \in W_j$ for $1 \le i, j \le s$ in the tree. We say that a group Steiner tree is *minimal* if it has no any group Steiner tree as a subgraph.

Next, we define some terminologies for directed graphs. Let $D = (V, E)$ be a directed graph. If D has an edge (u, v), v is called an *out-neighbor* of u and u is called an *in-neighbor* of v. Similarly, e is called an *incoming edge* of v and e is called an *outgoing edge* of u. We say that a vertex v is a *source* (resp. *sink*) if v has no in-neighbors (resp. no out-neighbors) in D. Let T be a subgraph of D. We say that T is a *directed tree rooted at r* if T has the unique source r and exactly one directed path from r to each vertex in T. A vertex ℓ in a directed tree is called a *leaf* if ℓ has no out-neighbors. Let W be a set of terminals and let $r \in V \setminus W$. We say that a subgraph H is a *directed Steiner subgraph* of (D, W, r) if H has a directed r-w path for any terminal $w \in W$. In particular, a directed Steiner subgraph T of (D, W, r) is a *directed Steiner tree* of (D, W, r) if T forms a directed tree rooted at r. A directed Steiner subgraph H is *minimal* if no proper subgraph of H is a directed Steiner subgraph of (D, W, r).

In this paper, in addition to STEINER TREE ENUMERATION, we address the following problems.

DEFINITION 4 (STEINER FOREST ENUMERATION). *Given an undirected graph $G = (V, E)$ and terminal sets $\mathcal{W} = \{W_i \subseteq V : 1 \le i \le s\}$, the problem asks to enumerate all the minimal Steiner forest of $(G, \mathcal{W})$.*

DEFINITION 5 (INTERNAL STEINER TREE ENUMERATION). *Given an undirected graph $G = (V, E)$ and a terminal set $W \subseteq V$, the problem asks to enumerate all the Steiner trees T of (G, W) in which every vertex in W belongs to T as an internal vertex.[2]*

DEFINITION 6 (TERMINAL STEINER TREE ENUMERATION). *Given an undirected graph $G = (V, E)$ and a terminal set $W \subseteq V$, the problem asks to enumerate all the minimal Steiner trees T of (G, W) in which every vertex in W belongs to T as a leaf vertex.*

DEFINITION 7 (DIRECTED STEINER TREE ENUMERATION). *Given a directed graph $D = (V, E)$, a root $r \in V$, and a terminal set $W \subseteq V \setminus \{r\}$, the problem asks to enumerate all the minimal subgraphs of D in which there is a directed path from r to $v \in W$.*

DEFINITION 8 (GROUP STEINER TREE ENUMERATION). *Given an undirected graph $G = (V, E)$ and terminal sets $\mathcal{W} = \{W_i \subseteq V : 1 \le i \le s\}$, the problem asks to enumerate all minimal group Steiner trees of $(G, \mathcal{W})$.*

As the following problem is slightly different from the other problems, we need further definitions. Let $W \subseteq V$ be a terminal set. A subgraph of G is called an *induced Steiner subgraph* of (G, W) if it is a Steiner subgraph of (G, W) and induced by some subset of vertices in G. An induced Steiner subgraph of (G, W) induced by

---

[2]In this definition, solutions are not required to be "minimal".

$U \subseteq V$ is *minimal* if $G[U']$ is not an induced Steiner subgraph of (G, W) for every proper subset U' of U.

DEFINITION 9 (INDUCED STEINER SUBGRAPH ENUMERATION). *Given an undirected graph $G = (V, E)$ and a terminal set W, the problem asks to enumerate all the minimal induced Steiner subgraphs of (G, W).*

3 DIRECTED s-t PATH ENUMERATION REVISITED

Enumeration of all s-t paths in undirected graphs is one of the most famous and classical enumeration problems, which is indeed a special case of STEINER TREE ENUMERATION. There are several efficient enumeration algorithms for this problem [3, 21, 29, 35]. Our proposed algorithms use an s-t path enumeration algorithm as a subroutine. Note that Read and Tarjan [29] gave an amortized $O(n + m)$ time and $O(n + m)$ space algorithm for enumerating s-t paths in undirected graphs. It is easy to extend this result to directed graphs and turn into an $O(n + m)$ delay bound. However, we cannot use their algorithm as a black box in STEINER TREE ENUMERATION since this may yield the space bound $O(|W| (n + m))$, where $|W|$ is the number of terminals. To reduce $O(|W| (n + m))$ space to $O(n + m)$ space, we need to dive into the details of their algorithm and modify it, which is described below.

Let $D = (V, E)$ be a directed graph and let $\mathcal{P}(s, t, D)$ be the set of all directed s-t paths in D. The idea of the algorithm is as follows. We initially compute an arbitrary path $Q = (v_1, \ldots, v_k)$ with $v_1 = s$ and $v_k = t$ in D and output it. Then, the entire enumeration problem $\mathcal{P}(s, t, D)$ can be partitioned into subproblems $\mathcal{P}_i$ for $1 \leq i < k$, each of which requires to enumerate all directed s-t paths that contains $(v_1, \ldots, v_i)$ as a subpath and does not contain edge (v_i, v_{i+1}). Since every path in $\mathcal{P}(s, t, D)$ is either Q or enumerated in exactly one of these subproblems $\mathcal{P}_i$, the algorithm correctly enumerates all the paths in $\mathcal{P}(s, t, D)$. To solve each subproblem $\mathcal{P}_i$, it suffices to enumerate all v_i-t paths in $D[V \setminus \{v_1, \ldots, v_{i-1}\}] - (v_i, v_{i+1})$.

The algorithm based on this branching strategy is given in Algorithm 1. We assume that for each vertex v, the outgoing edges incident to v are totally ordered. Given this, for two edges e, f incident to v, we denote by $e \prec_v f$ if e is smaller than f in this order and by $e \preceq_v f$ if either $e = f$ or $e \prec_v f$. The algorithm first enumerates directed paths $Q^0, Q^1, \ldots, Q^p$ starting from s such that the edges incident to s' in these paths are distinct. This can be done by F-STP with the main loop of E-STP. The subroutine F-STP(D', s', t, e, f) computes a directed s'-t path Q^j in D' that avoids e and all outgoing edges f' incident to s' with $f' \preceq_{s'} f$. As there can be multiple choices of these directed paths, we choose a path whose edge incident to s' is "smallest" among all possible such paths. This implies that if D' has an s'-t path containing an outgoing edge (s', v), there is also a path Q^j containing it, which is enumerated as above. Then, the algorithm enumerates subproblems for each path Q^j and makes recursive calls. On each recursive call, for a directed s-s' path P and a directed s'-t path Q^j, we output $P \circ Q^j$ as a solution, where $P \circ Q^j$ is the directed s-t path obtained by concatenating P and Q^j with connection s'.

To enumerate all directed s-t paths, call E-STP$((s), \perp, 0, t)$. In what follows, let $\mathcal{T}$ be the rooted tree generated by the execution of

Algorithm 1 whose root corresponds to E-STP$(((s), \perp, 0, t))$, that is, this tree structure has a node for each call of E-STP. We call $\mathcal{T}$ the *enumeration tree*.

LEMMA 10. *Algorithm 1 enumerates all directed s-t paths without duplication.*

PROOF. Let $\mathcal{P}(s, t, D)$ be a collection of directed s-t paths in D. Since every path outputted by Algorithm 1 is the concatenation of an s-s' path and an s'-t path, the algorithm only outputs an s-t path in D. Thus, we show that Algorithm 1 outputs all paths in $\mathcal{P}(s, t, D)$ without duplication.

We first prove that the algorithm outputs every path $R \in \mathcal{P}(s, t, D)$. Suppose for contradiction that a path $R = (u_1, \ldots, u_k)$ is not output by the algorithm. Let $P = (u_1, \ldots, u_{k'})$ be the maximal subpath of R such that E-STP(P, e, d, t) is called during the execution of the algorithm. Such a path P is well-defined since we initially call E-STP$((s), \perp, 0, t)$. The edge e does not belong to R due to the maximality of P. Moreover, as the main loop in E-STP with F-STP enumerates $u_{k'}$-t paths $Q^0, Q^1, \ldots, Q^p$, exactly one of them, say Q^j, uses edge $(u_{k'}, u_{k'+1})$ as the outgoing edge incident to $u_{k'}$. If $P \circ Q_j = R$, we are done. Otherwise, there is an edge $(u_{k''}, u_{k''+1})$ in Q^j that does not belong to R. We indeed call E-STP$(P \circ Q_i^j, (u_{k''}, u_{k''+1}), d + 1, t)$ for appropriate i, which contradicts the maximality.

We show that the output of the algorithm has no duplication. Let L_1 and L_2 be a pair of leaf nodes in the enumeration tree $\mathcal{T}$ and X be the lowest common ancestor of L_1 and L_2. The algorithm generates child nodes of X by adding distinct edges to the subpath on X. Thus, the solutions on L_1 and L_2 must be distinct and the algorithm has no duplication. □

We next consider the time complexity of the algorithm. We can easily confirm that F-STP runs in $O(n + m)$ time. For each path Q computed by F-STP, we output a solution $P \circ Q$. To achieve $O(n + m)$ delay, we need to enumerate all children in $O(n + m)$ delay at each node X in the enumeration tree $\mathcal{T}$. Let P be an s-s' path and $Q = (v_1, \ldots, v_k)$ be an s'-t path such that $P \circ Q$ is an s-t path in D. For each $2 \leq i \leq k$, we let $Q_i = (v_1, \ldots, v_i)$. If we can enumerate all subpaths Q_i such that $D[V \setminus (V(P \circ Q_i) \setminus \{v_i\})]$ contains a directed v_i-t path avoiding (v_i, v_{i+1}) in $O(n + m)$ delay, then we can enumerate all children in $O(n + m)$ delay as well. Given this, we say that Q_i is *extendible with P* if there is a directed v_i-t path in $D[V \setminus (V(P \circ Q_i) \setminus \{v_i\})] - (v_i, v_{i+1})$. The following lemma is crucial for efficiently obtaining such subpaths Q_i.

LEMMA 11. *Let Q_i be a subpath of Q that is extendible with P. Then, we can, in $O(n + m)$ time, either find the largest index i^* with $i^* < i$ such that Q_{i^*} is extendible with P or determine no such index exists.*

PROOF. Let $2 \leq j \leq i$. To compute the reachability from v_j to t in $D_j = D[V \setminus (V(P \circ Q_j) \setminus \{v_j\})] - (v_j, v_{j+1})$ efficiently, we use a boolean value $r(j, u)$ for each vertex u in D_j defined as follows[3]. We set $r(j, u)$ to true if and only if D_j has a directed path from u to t. For $j = i$, we can obtain all values $r(j, \cdot)$ in $O(n + m)$ time using a standard graph search algorithm.

---

[3]The role of $r(i, u)$ is the same as $d(\cdot)$ in [29] but we slightly modify the notion for the expository purpose.

Algorithm 1: A linear-delay enumeration algorithm for directed s-t path enumeration.

Input: P: a directed s-s' path, e: an edge cannot be used for extending P, d: depth of recursion.

1   **Procedure** E-STP(P, e, d, t)
2     $Q^0 = (v_1^0, \ldots, v_k^0) \leftarrow$
     F-STP$(D[V \setminus (V(P) \setminus \{s'\})], s', t, e, \bot)$
3     $j \leftarrow 0$
4   **while** $Q^j \neq \bot$ **do**
5   **if** d is even **then**   Output $P \circ Q^j$
6   **for** $i = k-1, \ldots, 2$ **do**
7   Let $Q_i^j = (v_1^j, \ldots, v_i^j)$ be a subpath of Q^j
8   **if** $D[V \setminus (V(P \circ Q_i^j) \setminus \{v_i^j\})] - (v_i^j, v_{i+1}^j)$ has a
         directed v_i^j-t path **then**
9   E-STP$(P \circ Q_i^j, (v_i^j, v_{i+1}^j), d+1, t)$
10   **if** d is odd **then**   Output $P \circ Q^j$
11       $Q^{j+1} = (v_1^{j+1}, \ldots, v_k^{j+1}) \leftarrow$
       F-STP$(D[V \setminus (V(P) \setminus \{s'\})], s', t, e, (v_1^j, v_2^j))$
12       $j \leftarrow j+1$
13 **Procedure** F-STP(D', s', t, e, f)
14   **if** $e \neq \bot$ **then** Remove e from D'
15   **if** $f \neq \bot$ **then**
16   Remove outgoing edges f' incident to s' with
      $f' \preceq_{s'} f$ from D'
17   Compute a directed s'-t path $Q = (u_1, \ldots, u_k)$ in D'
    such that there is no directed s'-t path
    $Q' = (u_1', \ldots, u_{k'}')$ with $(u_1', u_2') \prec_{s'} (u_1, u_2)$ in D
18   **if** no such path Q exists **then**   **return** $\bot$
19   **return** Q

We consider how to obtain $r(j, u)$ from $r(j+1, u)$ for $j < i$. Let V_j and E_j be the set of vertices and the set of edges in D_j, respectively. Note that $V_{j+1} \subseteq V_j$ and $E_{j+1} \subseteq E_j$. This implies that $r(j, u)$ is true, then $r(j', u)$ is also true for any $j' < j$.

We set the value of $r(j, v)$ for any vertex v in D_j as follows. If $v \notin V_{j+1}$, then $r(j+1, v)$ is false. Let F be the set consisting of edges (v, w) in $E_j \setminus E_{j+1}$ such that $r(j+1, v)$ is false and $r(j+1, w)$ is true. In the following, we use F as a program variable. For each edge (v, w) in F, we set $r(j, v)$ to true, remove (v, w) from F, and add all incoming edges (x, v) of v to F if $r(j+1, x)$ is false. We repeat this until F is empty.

We show the correctness of this update procedure. For any vertex u in D_j such that $r(j, u)$ is true, if $r(j+1, u)$ is false, then all the paths from u to t must contain at least one edge in the initial set of F. Let k be the shortest distance from u to the closest vertex w in D_j such that $r(j+1, w)$ is true. If $k = 1$, by following edges in the initial set of F, we can set the correct value to $r(j, u)$. In addition, if we add an edge to F, the tail is reachable to t. Thus, for any k, we can correctly update the values of r by the transitivity of the reachability to t.

Each edge in D is added to F at most once. Moreover, we maintain the reachability $r(\cdot, \cdot)$ in an array of length n since $r(j, u)$ is true

only if $r(j', u)$ is true for all $j' \leq j$. Thus, this procedure finds i^* in linear time and linear space. □

We next consider the space complexity of the algorithm. Let X be a node of $\mathcal{T}$ and P_X be a directed s-t path outputted in X. To reduce the space complexity of our algorithms for STEINER TREE ENUMERATION and variants, the important observation is that the total space for storing some information on each node for successive recursive calls is $O(|P_X|)$, see the appendix for the details. The algorithm can be applied to undirected graphs by simply replacing each undirected edge with two directed edges with opposite directions. We should remark that Birmelé *et al.* proposed an efficient enumeration algorithm for undirected s-t paths running in $O\left(\sum_{P \in \mathcal{P}(s,t,G)} |P|\right)$ total time [3].

THEOREM 12. *Algorithm 1 enumerates all s-t paths of a directed (undirected) graph in $O(n+m)$ delay and $O(n+m)$ space.*

PROOF. We first consider the space complexity. We traverse the enumeration tree in a depth-first manner and then need to store some information on each node for successive recursive calls. For each node X and its child Y whose associated instances are (P, e, d, t) and $(P \circ Q_i^j, (v_i^j, v_{i+1}^j), d+1, t)$, respectively, we store the subpath Q_i^j, the three directed edges e, (v_i^j, v_{i+1}^j), and f, and the loop variables i, j. Note that the entire path $P \circ Q_i^j$ and the depth d are maintained in a global memory. To compute the next sibling Z of Y, we need to restore path P and Q^j. Since the entire path $P \circ Q_i^j$ is stored in a global memory, we can compute P from Q_i^j in $O(n+m)$ time. From the restored path P and two directed edges $e = (v_i^j, v_{i+1}^j)$ and f, Q^j can be recomputed in $O(n+m)$ time as well since we use the fixed directed path finding algorithm F-STP. Paths Q_i^j are edge-disjoint at any nodes having the ancestor-descendant relation. Thus, the total space for storing information in successive recursive calls on a node X is $O(|P_X|)$ and the space complexity of the algorithm is $O(n+m)$, where P_X is a directed s-s' path on X.

We next discuss the delay of the algorithm. We can compute paths $Q^0, Q^1, \ldots, Q^p$ in $O(n+m)$ delay. For each path Q^j, by Lemma 11, we can compute the next sibling in $O(n+m)$ time. Moreover, we output a solution in a pre-order (resp. post-order) manner if the depth of the node is even (resp. odd)[4]. Since each node output at least one solution, we output at least one solution between three consecutive nodes in the depth-first search traversal of $\mathcal{T}$. Thus, the delay of the algorithm is $O(n+m)$. □

Algorithm 1 can be extended to the one for enumerating paths between two disjoint subsets S and T of V. Let D be a directed graph. We remove all edges directed from $v \in T$ and directed to $v \in S$. We then add a vertex s to D and an edge (s, v) for each $v \in S$. Similarly, we also add a vertex t to D and an edge (v, t) for each $v \in T$. Then, by enumerating all s-t paths and removing s and t from these paths, we obtain all S-T paths in the original graph.

---

[4]This technique is known as the alternating output method [33].

Algorithm 2: A polynomial-delay algorithm to enumerate all minimal Steiner trees. Note that the root recursive call is E-MST$(G, W, (\{w_0\}, \emptyset))$, where w_0 is an arbitrary terminal.

Input: G: a graph, W: a set of terminals, T: a partial Steiner tree of (G, W)

1 **Procedure** E-MST(G, W, T)
2   **if** T *is a minimal Steiner tree of* (G, W) **then** Output T
3   **else**
4   Let w be an arbitrary terminal with $w \notin V(T)$
5   **for** *each* $V(T)$-w *path* P *in* G **do**
6   E-MST$(G, W, T + E(P))$

---

4 ENUMERATING MINIMAL STEINER TREES VIA BRANCHING

In this section, we give efficient algorithms for STEINER TREE ENUMERATION by incorporating the path enumeration algorithm described in the previous section. To explain our idea clearly, we first give a simple polynomial-delay and polynomial-space enumeration algorithm. Then, we give a linear-delay algorithm with a non-trivial analysis.

4.1 Polynomial-delay enumeration of minimal Steiner trees

Let $G = (V, E)$ be an undirected graph and let $W \subseteq V$ be terminals. Our enumeration algorithm, given a graph G, $W \subseteq V$, and a subgraph T of G, enumerates minimal Steiner trees of (G, W) that contain T as a subgraph. In this section, we assume that G is connected. Moreover, during the execution of our algorithm, T always forms a tree whose leaves are all terminal. We call such a tree T a *partial Steiner tree of* (G, W). Note that every leaf of T is terminal but some terminals may not be contained in T. Then, we have the following easy but key lemma for partial Steiner trees.

LEMMA 13. *If* T *is a partial Steiner tree of* (G, W), *then there is a minimal Steiner tree of* (G, W) *that contains* T *as a subgraph.*

PROOF. Let T be a partial Steiner tree of (G, W). As G is connected, there is a spanning tree T' of (G, W) containing T as a subgraph. By greedily removing non-terminal leaves from T', we have a Steiner tree of (G, W) containing T as a subgraph whose leaves are all terminal, which is a minimal Steiner tree of (G, W) by Proposition 3. Moreover, as every leaf of T is terminal, T is contained in this minimal Steiner tree. □

Our polynomial-delay enumeration algorithm is shown in Algorithm 2. The algorithm is based on branching and defines a rooted tree whose root corresponds to the initial call E-MST$(G, W, (\{w_0\}, \emptyset))$, where w_0 is an arbitrary terminal in W. We call this rooted tree an *enumeration tree*. In each recursion step, fix a terminal $w \in W \setminus V(T)$ and generate new partial Steiner tree $T + E(P)$ for every $V(T)$-w path P in G. By Lemma 13, every partial Steiner tree generated by Algorithm 2 is a subgraph of some minimal Steiner tree of (G, W), assuming that G is connected. This implies that the algorithm outputs a minimal Steiner tree of (G, W) at every leaf node in the enumeration tree. The correctness of the algorithm is given in the following lemma.

LEMMA 14. *Algorithm 2 enumerates all minimal Steiner trees of* (G, W) *without duplication.*

PROOF. Let X be a node in the enumeration tree made by the algorithm with initial call E-MST$(G, W, (\{w_0\}, \emptyset))$ for some $w_0 \in W$. Suppose that the partial Steiner tree associated with X is T. Let $\mathcal{S}(X)$ be the set of minimal Steiner trees T^* of (G, W) associated with X that satisfy $E(T) \subseteq E(T^*) \subseteq E(G)$. First, we show that the algorithm outputs all minimal Steiner trees in $\mathcal{S}(X)$ at a leaf of the enumeration tree that is a descendant of X. Suppose that X is an internal node in the enumeration tree as otherwise, we are done. Let T^* be an arbitrary minimal Steiner tree in $\mathcal{S}(X)$. Let w be a terminal with $w \notin V(T)$. Since T^* is a Steiner tree of (G, W), there is a unique $V(T)$-w path P in T^*. Clearly, $T + E(P)$ is a partial Steiner tree of (G, W), there is a child of X whose associated instance is $(G, W, T + E(P))$ as we branch all possible $V(T)$-w paths in Algorithm 2. Since $T + E(P)$ is a subtree of T^*, by inductively applying this argument, T^* is output at some leaf node in the enumeration tree.

Next, we show that the algorithm does output all minimal Steiner trees of (G, W) without duplication. Suppose for contradiction that there is a minimal Steiner tree T^* of (G, W) that is output at two distinct leaf nodes L_1 and L_2 in the enumeration tree. Let X be the lowest common ancestor node of L_1 and L_2 and let T be the partial Steiner tree of (G, W) associated to node X. Let X_1 and X_2 be the children of X that are ancestor nodes of L_1 and L_2, respectively. As $X_1 \neq X_2$, the partial Steiner trees associated to X_1 and X_2 contain distinct $V(T)$-w path for some $w \in W \setminus V(T)$. This contradicts the uniqueness of the $V(T)$-w path in T^*. □

We run the path enumeration algorithm discussed in the previous section at Line 5 in Algorithm 2 and immediately call E-MST$(G, W, T + E(P))$ for each path P when it is output by the path enumeration algorithm. Thus, we have the next theorem for the complexity of Algorithm 2.

THEOREM 15. *Algorithm 2 enumerates all minimal Steiner trees of* (G, W) *in* $O(|W|(n + m))$ *delay and* $O(|W|(n + m))$ *space, where* n *and* m *are the number of vertices and edges of* G, *respectively.*

PROOF. By using an $O(n + m)$ delay algorithm for enumerating all s-t paths in Section 3, we can enumerate in $O(n + m)$ delay all children of a node in the enumeration tree. Moreover, the depth of the enumeration tree is at most $|W|$. Hence, the delay of our proposed algorithm is $O(|W|(n + m))$. It is sufficient to store the information of recursive calls for backtracking with $O(n + m)$ space per node. Therefore, the algorithm runs in $O(|W|(n + m))$ space in total. □

4.2 Improving the delay and space bound

In this subsection, we show that we can enumerate all the minimal Steiner trees of (G, W) in $O(n + m)$ delay and $O(n^2)$ space with $O(nm)$ preprocessing time. The main obstacle for achieving $O(n + m)$-delay enumeration is that some internal node in the enumeration tree obtained by Algorithm 2 may have only one child. Although each child is generated in $O(n + m)$ time, it is hard to enumerate solutions in linear delay if the number of internal nodes is much larger than the number of leaves. Hence, we need to modify

Line 4 in the algorithm so that each internal node in the enumeration tree has at least two children, which implies that the number of internal nodes is at most the number of leaves.

To improve the running time bound, we first give an algorithm with amortized $O(n + m)$ time per solution. Suppose that every internal node has more than one child node in the enumeration tree defined in the previous section. The total running time of the algorithm is upper bounded by $\sum_X O(n + m) \cdot |ch(X)|$, where the summation is taken over all nodes X in the enumeration tree and $ch(X)$ is the set of children of node X. Since the number of internal nodes is at most the number of leaf nodes and the algorithm outputs exactly one solution at each leaf node, the running time of the algorithm is amortized $O(n + m)$ time. Moreover, as we will discuss at the end of this subsection, we can prove that (a modified version of) this algorithm runs in $O(n + m)$ delay by means of the output queue method due to Uno [33].

We first "improve" the enumeration tree discussed in the previous subsection. The following lemma is a key to this improvement.

Lemma 16. *Let T be a partial Steiner tree of (G, W) and let $w \in W \setminus V(T)$. Let P be a $V(T)$-w path in G. Then, P is the unique $V(T)$-w path in G if and only if all the edges in P are bridges in G.*

Proof. Suppose that there are at least two $V(T)$-w paths, say P and P', in G. Then, there is a closed walk passing through P, P', and some part of T, implying that G has a cycle containing at least one edge from P, implying that this edge is not a bridge in G.

Conversely, suppose there is an edge e in P that is not a bridge of G. Let P' be a path between the end vertices of e in $G - e$. If $V(P') \cap V(T) = \emptyset$, $P - e + E(P')$ is a $V(T)$-w path distinct from P, which implies that there are at least two $V(T)$-w paths in G. Otherwise, we can still find a $V(T)$-w path that is a subpath of P', and hence the lemma follows. □

By Lemma 16, using a linear-time bridge enumeration algorithm [32], we can either find a terminal $w \notin V(T)$ such that G has at least two $V(T)$-w paths or conclude that there is no such a terminal in linear time. To do this, we first compute a minimal Steiner tree T' of (G, W) that contains T as a subgraph. By Proposition 3, this can be done in $O(n + m)$ time by simply finding a spanning tree that contains T and then removing redundant non-terminal leaves. Then, we can check whether each edge in $E(T') \setminus E(T)$ is a bridge in G. If there is a $V(T)$-w path in T' for some $w \in V(T) \setminus W$ that contains a non-bridge edge in G, by Lemma 16, we can conclude that there are at least two $V(T)$-w paths in G. Otherwise, there is no such a terminal, T' is indeed the unique minimal Steiner tree of (G, W) containing T as a subgraph. In this case, the node associated with (G, W, T) can be considered as a leaf node by modifying the enumeration tree and the unique minimal Steiner tree T' can be computed in $O(n + m)$ time. Based on these two cases, we can assume that each internal node of the enumeration tree has at least two children. We call the enumeration tree defined by this modification the *improved* enumeration tree.

Theorem 17. *There is an algorithm that enumerates all minimal Steiner trees of (G, W) in $O(n + m)$ amortized time per solution with $O(n + m)$ space.*

Figure 1: The structure of the improved enumeration tree. The thick line illustrates the path $\mathcal{P}$ between the root and node S at which we find the n-th solution in the preprocessing phase.

Proof. Since we can enumerate all $V(T)$-w paths in $O(n + m)$ delay and each internal node has at least two children, the total running time of the algorithm is upper bounded by $O((n + m)N)$, where N is the number of minimal Steiner trees of (G, W), which yields the claimed running time bound.

We next consider the space complexity. We should note that G, W, and T are stored as global data structures: We do not make copies on each recursive call. T is maintained as edge-disjoint paths in the data structure. For each node in the enumeration tree, we store a $V(T)$-w path P of G in the global data structure when calling E-MST$(G, W, T + E(P))$ and overwrite P with P' when calling E-MST$(G, W, T + E(P'))$ for the next $V(T)$-w path P'. Thus, we can maintain partial Steiner tree T in $O(n + m)$ space. To estimate the entire space consumption, consider a node X associated to instance (G, W, T) in the enumeration tree $\mathcal{T}$ of Steiner Tree Enumeration. For each path P output by the s-t path enumeration algorithm, we use $O(|P|)$ space, see Section 3 for the details. Since the total length of paths is equal to the size of a minimal Steiner tree, the algorithm runs in $O(n + m)$ space. □

To obtain a linear-delay bound, we employ the output queue method by Uno [33]. Roughly speaking, since the algorithm traverses the enumeration tree in a depth-first manner, we can see this as an Eulerian tour starting and ending at the root node in the graph obtained by replacing each edge with two parallel edges. The time elapsed between two adjacent nodes in this tour is upper bounded by $O(n + m)$. However, as the algorithm outputs solutions at leaf nodes only, the delay can be $\Omega(|W|(n + m))$. The idea of this technique is that we use a buffer of solutions during the traversal. Since the number of leaf nodes is larger than that of internal nodes, we can "periodically" output a solution using this buffer.

We slightly modify the original method in [33] to fit our purpose. Let X be a node in the improved enumeration tree $\mathcal{T}$ made by our algorithm. We say that X is *discovered* when it is visited for the first time and is *examined* when either the parent of X is visited after visiting X for the last time or the algorithm halts. We may identify these two cases for each leaf node X since they must occur consecutively. Firstly, we find the first n solutions and add them into a queue Q without outputting these solutions. Let $\mathcal{T}_{\text{pre}}$ be the subtree of $\mathcal{T}$ induced by the nodes that are discovered in the preprocessing phase. Secondly, after this preprocessing phase, we output solutions conforming to some rules. Let $\mathcal{T}_1, \ldots, \mathcal{T}_\ell$ be the connected components of $\mathcal{T} - V(\mathcal{T}_{\text{pre}})$, which are indeed subtrees of $\mathcal{T}$ rooted at some undiscovered nodes (See Figure 1). We order

these subtrees in such a way that for $1 \le i < j \le \ell$, the root of $\mathcal{T}_i$ is discovered before discovering that of $\mathcal{T}_j$ during the execution of the algorithm.

To bound the delay of the algorithm, we first discuss the strategy for outputting solutions from Q in $\mathcal{T}_i$. When we traverse $\mathcal{T}_i$, we output a solution at X from Q according to the following rules:

R1. output a solution from Q if X is an internal node, the depth of X is odd in $\mathcal{T}_i$, and X is examined,

R2. output a solution from Q if X is an internal node, the depth X is even in $\mathcal{T}_i$, and X is discovered,

R3. output the solution found on X if X is a leaf of $\mathcal{T}_i$ and either Q contains $3n/2$ solutions or X is the third or subsequent leaf child of the parent of X (meaning that there are at least two leaf siblings of X that have been already examined).

For a leaf node X, we add a solution that is found on X to Q without outputting it whenever **R3** does not hold.

The entire algorithm is described as follows. In the preprocessing phase, we compute n solutions by traversing the improved enumeration tree. Recall that $\mathcal{T}_{\text{pre}}$ is the subtree induced by the nodes discovered in this phase. Observe that the number of nodes in $\mathcal{T}_{\text{pre}}$ is $O(n)$. To see this, consider the path $\mathcal{P}$ between the root and the leaf node S at which we find the n-th solution in $\mathcal{T}$ (see Figure 1). Clearly, $\mathcal{P}$ has at most n nodes. Moreover, as each connected component in $\mathcal{T}_{\text{pre}} - \mathcal{P}$ is a rooted tree whose internal nodes have at least two children, $\mathcal{T}_{\text{pre}} - \mathcal{P}$ contains $O(n)$ nodes. After the preprocessing phase, we proceed with the traversal from S to find the root of $\mathcal{T}_1$ that is the first discovered node after the preprocessing phase. During the traversal in $\mathcal{T}_i$, for $1 \le i \le \ell$, we output solutions as above. Finally, during the traversal from S to each root of $\mathcal{T}_i$ in $\mathcal{P}$, we output a solution from Q at each odd depth node when it is examined.

For the sake of simplicity, we assume that n is even. To prove the correctness of the algorithm, we first show that, for each $\mathcal{T}_i$, the algorithm successfully outputs solutions from Q according to the rules **R1**, **R2**, and **R3**, assuming that Q contains $r \ge n/2$ solutions when the root of $\mathcal{T}_i$ is discovered. Moreover, when the root of $\mathcal{T}_i$ is examined, Q contains at least r solutions, assuming that $n/2 \le r \le n$. Fix $\mathcal{T}_i$. Let I and L be the sets of internal nodes and leaf nodes in $\mathcal{T}_i$, respectively. We define a function $\theta : I \to L$ that satisfies the following conditions: θ is injective, $\theta(X)$ is a descendant of X for every $X \in I$, and for $Y \in L$ if Y is the third or subsequent leaf child of a node X, then $\theta^{-1}(Y)$ is undefined. We say that a function θ is *consistent* if it satisfies these conditions. In our proof, we will keep track of solutions in Q based on a consistent function.

LEMMA 18. *Let T be a rooted tree such that every internal node has at least two children. Then, there is a consistent function from the internal nodes to the leaf nodes in T.*

PROOF. Without loss of generality, we assume that each node in T has at most two leaves as its children. We prove the lemma by giving an algorithm to construct a consistent function θ. The algorithm traverses T in a depth-first manner. When we reach a leaf ℓ in T, we define $\theta(x) = \ell$, where x is the nearest ancestor of ℓ such that $\theta(x)$ has not been defined so far. If there is no such node x, we do nothing at ℓ. Now, we claim that this function θ is consistent. It is easy to see that for every internal node x, $\theta(x)$ is a

descendant of x if $\theta(x)$ is defined. Thus, we prove that $\theta(x)$ is well-defined for every internal node x in T. Suppose for contradiction that there is an internal node x such that $\theta(x)$ is not defined in the above algorithm. Assume that x is a deepest node satisfying this condition. Let y be an arbitrary child of x and let T_y be the subtree of T rooted at y. Since $\theta(y)$ is defined and the number of internal nodes is strictly smaller than that of leaves in T_y, there is a leaf node ℓ that is not assigned to every internal node in T_y. By the construction of θ, $\theta(x) = \ell$, contradicting to the assumption. □

By the assumption, Q contains at least r solutions for some $n/2 \le r \le n$. We call arbitrary $n/2$ of them the *initial solutions* and the remaining $r - n/2$ of them the *excess solutions*. In the following, we abuse the queue Q as follows. Before starting the transversal of $\mathcal{T}_i$, we replace each initial solution S in Q with a pair (S, d) for some integer d with $0 \le d \le n - 1$ in such a way that Q contains a pair (S, d) for every even d with $0 \le d \le n - 1$. The value d is used for proving the correctness of the algorithm. This replacement can be done as Q contains at least $n/2$ solutions. We also replace each excess solution S with a pair $(S, *)$. In fact, we do not output these excess solutions during the traversal in $\mathcal{T}_i$. For each node X in $\mathcal{T}_i$, we denote by $d(X)$ the depth of X in $\mathcal{T}_i$. Then, the rules **R1**, **R2**, and **R3** can be interpreted as the following modified rules:

R'1. output a solution S of the form $(S, d(X))$ in Q if X is an internal node, $d(X)$ is odd, and X is examined,

R'2. output a solution S of the form $(S, d(X))$ in Q if X is an internal node, $d(X)$ is even, and X is discovered,

R'3. output the solution S found on X if X is a leaf of $\mathcal{T}_i$ and $\theta^{-1}(X)$ is undefined. If X is a leaf, $\theta^{-1}(X)$ is defined, and Q contains $3n/2$ solutions, we do the following. If Q contains a solution S' of the form $(S', d(\theta^{-1}(X)))$, then output S. Otherwise, as $r \le n/2$, there are two solutions S' and S'' such that Q contains both (S', d) and (S'', d) for some d. We replace (S'', d) with $(S, d(\theta^{-1}(X)))$ and output S''.

For leaf node X, we add the solution S found on X to $(S, d(\theta^{-1}(X)))$ without outputting it if Q contains less than $3n/2$ solutions. Then, the following proposition holds.

PROPOSITION 19. *Suppose that X is a leaf in $\mathcal{T}_i$ and $\theta^{-1}(X)$ is defined, that is, X is either the first or second leaf child of its parent. Then, Q contains a pair $(S, d(\theta^{-1}(X)))$, where S is the solution found at X, when X is examined. Moreover, if Q contains a pair (S', d) for some $0 \le d \le n - 1$ before discovering X, then it also contains a pair (S'', d) after examining X.*

Observe that these modified rules simulate the original rules: The timing of outputting solutions according to these modified rules are exactly the same with that for the original rules. Thus, if we can successfully output solutions according to the modified rules, we do so according to the original rules.

Now, we show that we can output solutions according to the modified rules. Let X be an internal node in $\mathcal{T}_i$. Suppose **R'1** occurs at X. Then, Q must contain a solution S of the form $(S, d(X))$ since the leaf $\theta(X)$ is already visited before examining X due to the consistency of θ. Note that although the solution found at $\theta(X)$ may be different from S, by Proposition 19, Q contains at least one pair $(S, d(X))$. Suppose next that **R'2** occurs at X. If X is the first depth-$d(X)$ node among all depth-$d(X)$ nodes in $\mathcal{T}_i$, as $d(X)$ is even,

Q contains an initial solution S of the form $(S, d(X))$. Otherwise, X is the second or subsequent depth-$d(X)$ node in $\mathcal{T}_i$. Then, Q contains a solution S of the form $(S, d(X))$ since $\theta(Y)$ is already visited at this point, where Y is the depth-$d(X)$ node that is examined immediately before X. Hence, we can successfully output solutions according to the modified rules.

Recall that we assume $n/2 \le r \le n$. We claim that after traversing $\mathcal{T}_i$, Q contains at least r solutions.

We observe that after traversing $\mathcal{T}_i$, Q contains a pair (S, d) for every even d with $0 \le d < n$. If d is larger than the height of $\mathcal{T}_i$, the initial solution S of the form (S, d) that is still in Q. Otherwise, let X be the depth-d node that is discovered for the last time. Then, the solution obtained at $\theta(X)$ is not output in the remaining traversal. Thus, Q contains a pair (S, d) for every even d with $0 \le d < n$. Moreover, we do not output any excess solutions in $\mathcal{T}_i$. Therefore, Q contains at least r solutions.

Theorem 20. *Steiner Tree Enumeration can be solved in $O(n + m)$ delay and $O(n^2)$ space using $O(nm)$ preprocessing time.*

Proof of Theorem 20. Since we can generate each child in time $O(n + m)$, we can move to the next node in the Eulerian tour in time $O(n + m)$ as well. Since $\mathcal{T}_{\mathtt{pre}}$ contains $O(n)$ node, the preprocessing phase is done in $O(nm)$ time. Let S be the leaf node at which the n-th solution is found and let $\mathcal{P}$ be the path between S and the root of $\mathcal{T}$. From the node S to the root of $\mathcal{T}_1$, we output a solution at each odd depth node. This implies that the delay in this time interval is also $O(n + m)$, and also for each time interval between two roots of $\mathcal{T}_i$ and $\mathcal{T}_{i+1}$ for $1 \le i < \ell$. We output at most $n/2$ solutions at nodes on $\mathcal{P}$. Moreover, if Q contains r ($n/2 \le r \le n$) solutions before traversing $\mathcal{T}_i$, then it contains at least r solutions after traversing $\mathcal{T}_i$. This implies that we can output solutions using the aforementioned strategy.

To bound the delay in the exploration in $\mathcal{T}_i$, we consider the sequence of internal nodes obtained from the Eulerian tour in $\mathcal{T}_i$ by removing leaf nodes. Let $(X_1, \ldots, X_{10})$ be an arbitrary consecutive subsequence of nodes in the sequence. Let us note that as the sequence is obtained from the Eulerian tour, these nodes may not be distinct. We show that at least one solution is output at those nodes. If all nodes between X_j to X_{j+3} are identical for some $1 \le j \le 7$, then X_j has at least three leaf nodes as its children and we output a solution at one of these children. Thus, we assume that the sequence contains at most three consecutive identical nodes. If there are only two distinct internal nodes in this sequence, these two nodes appear in an interleaved way, which is impossible in the Eulerian tour obtained from a rooted tree. Thus, we can assume that the sequence has at least three internal nodes. Let X_{j_1}, X_{j_2}, and X_{j_3} be three distinct internal nodes such that there is no other internal node between them in the sequence and $1 \le j_1 < j_2 < j_3 \le 10$. If all depths of these nodes are distinct, then we output a solution as there is either an odd depth node that is examined (**R1**) or an even depth node that is discovered (**R2**). Otherwise, the depth of X_{j_1} and X_{j_3} are identical. Since we examine X_{j_1} and discover X_{j_3}, we output a solution. Finally, we consider a minimal consecutive subsequence of nodes in the original Eulerian tour that contains $(X_1, \ldots, X_{10})$ as a subsequence. Since leaf nodes do not appear consecutively in this sequence, the length of the sequence is at most 20 and we can

Algorithm 3: A general description of enumeration algorithms for Steiner problems.

Input: G: a graph, $\mathcal{W}$: a (family of) terminal set(s), F: a partial solution in G

1 **Procedure** E-Sol$(G, \mathcal{W}, F)$
2   **if** *F is a solution* **then** Output F
3   **else**
4   Let W be a terminal (set) in $\mathcal{W}$
5   **for** *each valid path P for (F, W)* **do**
6   E-MST$(G, \mathcal{W}, F + E(P))$

---

output at least one solution in this sequence. Therefore, the delay of this algorithm is $O(n + m)$.

Finally, we consider the space complexity of the algorithm. Since Q stores the first n solutions, we use $O(n^2)$ space for Q. The remaining estimation of the space complexity is analogous to Theorem 17. Thus, the overall space complexity is $O(n^2)$. □

We note that this technique can be used when each internal node in the enumeration tree has at least two children. Thus, we also use this technique to obtain linear delay bounds for other problems, which will be discussed in the next section.

5 VARIANTS OF MINIMAL STEINER TREES

A similar branching strategy works for other variants of Steiner Tree Enumeration, such as Steiner Forest Enumeration, Terminal Steiner Tree Enumeration, and Directed Steiner Tree Enumeration. In addition, we can obtain linear-delay enumeration algorithms for these problems.

A general form of our algorithm is described as Algorithm 3. We are given an instance $(G, \mathcal{W}, F)$, where $\mathcal{W}$ is a (family of) terminal set(s), and F is a *partial solution*. Similarly to Algorithm 2, the execution of the algorithm defines a rooted tree whose nodes are associated to triple $(G, \mathcal{W}, F)$. We again call this rooted tree an *enumeration tree*. The algorithm recursively extends F into a larger partial solution F_P by adding a *valid path* P for F and a selected terminal (set) W, which can be computed by the s-t path enumeration algorithm. In this section, we focus on Steiner Forest Enumeration as a concrete example of these problems.

To specify Algorithm 3 for Steiner Forest Enumeration, we define partial solutions and their valid paths. Recall that for an undirected graph $G = (V, E)$ and a family of terminal sets $\mathcal{W} = \{W_1, \ldots, W_s\}$ with $W_i \subseteq V$, a forest F is a *Steiner forest of* $(G, \mathcal{W})$ if for any set W_i, F contains a path between every pair of vertices in W_i. In particular, F is a *minimal Steiner forest of* $(G, \mathcal{W})$ if there is no proper subgraph of F that is a Steiner forest of $(G, \mathcal{W})$. Note that when $|\mathcal{W}| = 1$, Steiner Forest Enumeration is equivalent to Steiner Tree Enumeration. We assume that each terminal set is contained in a connected component of G as otherwise, there is no minimal Steiner forest of $(G, \mathcal{W})$.

In what follows, without loss of generality, we can assume that $|W_i| = 2$ for each W_i. This follows from the observation that if there is a terminal set $W \in \mathcal{W}$ such that $W = \{w_1, \ldots, w_k\}$ with $k \ge 3$, every Steiner forest of $(G, \mathcal{W})$ is a Steiner forest of $(G, \mathcal{W}')$ with $\mathcal{W}' = (\mathcal{W} \setminus \{W\}) \cup \{\{w_1, w_2\}, \{w_1, w_3\}, \ldots, \{w_1, w_k\}\}$ and vice

versa. Also, from this observation, we can assume that the number of sets in $\mathcal{W}$ is at most $n - 1$. We note that by this conversion, terminal sets in $\mathcal{W}$ may intersect. For each $1 \leq i \leq s$, we let $W_i = \{w_i, w_i'\}$.

The next lemma plays an important role for Steiner Forest Enumeration, which enables us to enumerate all the solutions by combining paths, each of which connects a pair of terminals.

Lemma 21. F is a minimal Steiner forest of $(G, \mathcal{W})$ if and only if F is a forest consisting of the union of paths $P_1, \ldots P_s$ connecting $\{w_1, w_1'\}, \ldots, \{w_s, w_s'\}$, respectively. Moreover, let $\mathcal{P} = (P_1, \ldots, P_s)$ and $\mathcal{P}' = (P_1', \ldots, P_s')$ be two (ordered) sets of w_i-w_i' paths P_i and P_i' whose unions are minimal Steiner forests F and F' of $(G, \mathcal{W})$, respectively. Then, $\mathcal{P} \neq \mathcal{P}'$ if and only if $F \neq F'$.

Proof. Let F be a forest such that F consists of the union of paths P_i between w_i and w_i' for $1 \leq i \leq s$. Clearly, F is a Steiner forest of $(G, \mathcal{W})$. Moreover, for any edge $e \in E(P_i)$, $F - e$ has no w_i-w_i' path as otherwise F has a cycle. Thus, F is a minimal Steiner forest of $(G, \mathcal{W})$.

Conversely, let F be a minimal Steiner forest of $(G, \mathcal{W})$. Then, there is a unique path P_i between w_i and w_i' for each $1 \leq i \leq s$. Thus, F contains $\bigcup_{1 \leq i \leq s} E(P_i)$. Moreover, if F contains an edge $e \in F \setminus \bigcup_{1 \leq i \leq s} E(P_i)$, then this contradicts to the minimality of F. Hence, we have $F = \bigcup_{1 \leq i \leq s} E(P_i)$.

Clearly, $F \neq F'$ implies $\mathcal{P} \neq \mathcal{P}'$ as $F = \bigcup_{1 \leq i \leq s} E(P_i)$ and $F' = \bigcup_{1 \leq i \leq s} E(P_i')$. Let $\mathcal{P} = (P_1, \ldots, P_s)$ and $\mathcal{P}' = (P_1', \ldots, P_s')$ be as in the statement with $\mathcal{P} \neq \mathcal{P}'$. Then, there is a pair of w_i-w_i' paths P_i and P_i' with $P_i \neq P_i'$. Suppose that $F = F'$. Since $P_i \neq P_i'$, there are two paths between w_i and w_i' in F, which implies that F contains a cycle, a contradiction. □

Now, we define a partial solution and a valid path for Steiner Forest Enumeration. A forest F of G is called a *partial solution* or, more specifically, a *partial Steiner forest* of $(G, \mathcal{W})$ if F is a forest of the form $F = \bigcup_{\{w_i, w_i'\} \in \mathcal{W}'} E(P_i)$, where P_i is a w_i-w_i' path and $\mathcal{W}'$ is a subset of $\mathcal{W}$. Let F be a partial Steiner forest of $(G, \mathcal{W})$ and let $W = \{w, w'\} \in \mathcal{W}$ be a terminal pair such that there is no path between the pair in F. We say that a w-w' path P in G is *valid for* (F, W) if $F + E(P)$ has no cycles. Let us note that valid path P may contain edges in F. Clearly, $F + E(P)$ is a partial Steiner forest of $(G, \mathcal{W})$ as well.

Lemma 22. *If F is a partial Steiner forest of $(G, \mathcal{W})$, then there is a minimal Steiner forest of $(G, \mathcal{W})$ that contains F as a subgraph.*

Proof. Let F' be a maximal forest in G that contains F as a subgraph. By the assumption that every terminal set W_i is contained in a connected component of G, F' is a Steiner forest of $(G, \mathcal{W})$. From F', we repeatedly remove $e \in E(F') \setminus E(F)$ when $F' - e$ is a Steiner forest of $(G, \mathcal{W})$. Then, we let F^* be the Steiner forest obtained in this way. We show that F^* is a minimal Steiner forest of $(G, \mathcal{W})$.

Suppose that F^* contains an edge e satisfying that $F^* - e$ is a minimal Steiner forest of $(G, \mathcal{W})$. By the definition of F^*, it holds that $e \in F$. Since F is a partial Steiner forest, e belongs to a w-w' path in F for some $\{w, w'\} \in \mathcal{W}$. If $F - e$ has a w-w' path, then F has a cycle, which contradicts to the fact that F is a forest. □

The complete description of Algorithm 3 for Steiner Forest Enumeration is as follows. We initially call E-Sol$(G, \mathcal{W}, (\{w\}, \emptyset))$, where w is an arbitrary terminal in a terminal set. Clearly, $(\{w\}, \emptyset)$ is a partial Steiner forest of $(G, \mathcal{W})$. Let F be a partial Steiner forest that is not a Steiner forest of $(G, \mathcal{W})$ and let $W = \{w, w'\} \in \mathcal{W}$ be a terminal pair that is not connected in F. To enumerate all valid paths between w and w' for (F, W), we enumerate all w-w' paths P in $G/E(F)$. Since there is a one-to-one correspondence between $E(G) \setminus F$ and $E(G/E(F))$, $F + E(P)$ can be seen as a subgraph of G. It is easy to observe that $F + E(P)$ has no cycles, and then it is a partial Steiner forest of $(G, \mathcal{W})$, implying that the unique w-w' path in $F + E(P)$ is a valid path for (F, W). The next theorem shows the correctness and running time analysis of the above algorithm.

Theorem 23. Steiner Forest Enumeration *can be solved in $O(t(n + m))$ delay and $O(n + m)$ space, where $t = \left| \bigcup_{W \in \mathcal{W}} W \right|$.*

Proof. We first show that the algorithm enumerates all the minimal Steiner forests of $(G, \mathcal{W})$. Let F^* be a minimal Steiner forest of $(G, \mathcal{W})$ and F be a partial Steiner forest that is strictly contained in F^*. By the minimality of F^*, there is a terminal pair $W = \{w, w'\}$ that is not connected in F. Let P be the unique path between w and w' in F^*. Then, $F + E(P)$ has no cycles, which implies that P is a valid path for (F, W). By inductively applying the same argument to $F + E(P)$, we can eventually compute F^*.

Next, we show that the algorithm outputs minimal Steiner forests without duplication. Suppose for contradiction that there is a minimal Steiner forest F^* of $(G, \mathcal{W})$ that is output at two leaf nodes L_1 and L_2 in the enumeration tree. Let X be the lowest common ancestor of L_1 and L_2 and let F be the partial Steiner forest associated with X. Similarly to Lemma 14, the two children of X that are ancestors of L_1 and L_2 are associated to distinct partial Steiner forests $F + E(P_1)$ and $F + E(P_2)$ for some distinct valid paths P_1 and P_2 for (F, W) with some terminal set W. However, by Lemma 21, any minimal Steiner forests that respectively contain $F + E(P_1)$ and $F + E(P_2)$ must be distinct, contradicting to the assumption.

We finally analyze the delay and the space complexity. Note that the height of the enumeration tree obtained by our algorithm is at most n. Hence, since we use an $O(n + m)$ delay s-t path enumeration algorithm, the delay of our algorithm is $O(t(n + m))$. At each internal node, we store exactly one valid path for a partial Steiner forest F and, by an analogous argument in Theorem 17, the space complexity is $O(n + m)$. □

In the remaining of this section, we improve the delay complexity of the algorithm in the above theorem with a polynomial-time preprocessing phase as in Theorem 20. To this end, we need to ensure that each internal node of the enumeration tree has at least two children. Let X be an internal node in the enumeration tree and let $(G, \mathcal{W}, F)$ be the instance associated with X. Let $W = \{w, w'\} \in \mathcal{W}$ be a terminal set such that there is no path between w and w' in F. Then, there are at least two valid paths for (F, W) if and only if X has at least two children in the enumeration tree. By the one-to-one correspondence between $E \setminus E(F)$ and $E(G/E(F))$, from every w-w' path P in $G/E(F)$, we can obtain a unique valid path for (F, W). We note that the graph $G/E(F)$ may contain multiedges between two vertices, and then they are not considered as bridges

even if removing these edges increases the number of connected components of G. This correspondence gives the following lemma.

Lemma 24. *Let F be a partial Steiner forest of $(G, \mathcal{W})$ that has no w-w' path for some terminal pair $W = \{w, w'\} \in \mathcal{W}$. Let P be a valid path for (F, W). Then, P is the unique valid path for (F, W) in G if and only if every edge in $E(P) \setminus E(F)$ is a bridge in $G/E(F)$.*

Proof. We first show the if direction. To prove the uniqueness, suppose that there is another valid path $P' \neq P$ for (F, W). As F is a forest, we have $E(P) \setminus E(F) \neq E(P') \setminus E(F)$. This implies that there are two distinct paths between w and w' in $G/E(F)$, contradicting to the fact that every edge on P^* is a bridge in $G/E(F)$.

We next show the other direction. Suppose that P is the unique valid path for (F, W) in G. Since P is valid, $F + E(P)$ is a forest, and then $P' = P/E(F)$ is a w-w' path in $G/E(F)$. For contradiction, suppose that P' contains a non-bridge edge e in $G/E(F)$. Then, there is another w-w' path P'' in $G/E(F)$ that does not contain e. Since $F + E(P'')$ is also a partial Steiner forest, the unique path $F + E(P'')$ is a valid path for (F, W), contradicting to the uniqueness of P. Hence, the statement holds. □

By Lemma 24, we can determine whether there exists a pair $W \in \mathcal{W}$ such that there are at least two valid paths for (F, W) in linear time as follows. We could not directly apply a similar idea used in Steiner Tree Enumeration to Steiner Forest Enumeration as it is not obvious to compute a minimal Steiner forest of $(G, \mathcal{W})$ that contains F as a subgraph in linear time. First, we compute $G' = G/E(F)$ in linear time. Then, we find the set B of bridges in G' by a linear-time bridge enumeration algorithm [32], and obtain $G'' = G'/B$. By Lemma 24, there is a unique valid path for (F, W) in G if and only if two terminals in W are identical in G''. Thus, such a pair can be found in $O(n + m)$ time if it exists.

Next, we consider the other case, that is, for every terminal set $W \in \mathcal{W}$ there is exactly one valid path P for (F, W). In this case, there is a unique minimal Steiner forest F' of $(G, \mathcal{W})$ that contains F as a subgraph. By Lemma 24, $E(P) \setminus E(F)$ is composed of bridges in $G' = G/E(F)$. Let B be the bridges in G'. From the assumption, any terminal pair $\{w, w'\} \in \mathcal{W}$ is connected in $F + B$. Moreover, $F + B$ forms a forest as B consists of bridges in G'. However, $F + B$ may not be a minimal Steiner forest of (G, W). Thus, to obtain the unique minimal Steiner forest F' from $F + B$, we need to remove redundant bridges in B. Since we can independently remove redundant bridges from each connected component in $F + B$, we assume that $F + B$ forms a tree and all leaves are terminal.

The naive approach to obtain the unique F' is that we enumerate paths between each terminal pair in $\mathcal{W}$ and take the union of them. This approach runs in $O(tn)$ time, where $t = \left| \bigcup_{W \in \mathcal{W}} W \right|$. Thus, we employ another approach in which we exploit the lowest common ancestor (LCA, for short) of each terminal pair. Let $T = F + B$. We assume that T is a rooted tree by choosing an arbitrary vertex as its root. For any pair of vertices u and v in T, we denote by $lca(u, v)$ the LCA of u and v in T. The u-v path in T can be decomposed into $P(u, lca(u, v))$ and $P(lca(u, v), v)$, where $P(x, y)$ be the unique x-y path in T. To compute the LCA for pairs of terminals efficiently, we use a data structure due to Harel and Tarjan [16]. This data structure can be constructed in $O(n)$ time using $O(n)$ space and allows us to compute $lca(u, v)$ in $O(1)$ time for given u and v. Thus,

for each terminal pair $\{w, w'\}$ in T, we can find the w-w' path in $O(|E(P(w, w'))|)$ time. However, the running time is still quadratic to find all paths between terminal pairs.

To compute the set of edges in T that belong to at least one path between a pair of terminals, we make pairs $(lca(w, w'), w)$ and $(lca(w, w'), w')$ for every terminal pair $\{w, w'\}$ and sort them in the descending order of the height of $lca(w, w')$. Since there are at most $2n$ pairs and $lca(w, w')$ is an integer between 1 and n, we can sort them in $O(n)$ time. Then, for each pair (a, w) processed in this order, we mark all the edges on the path starting from w to its ancestor a and stop marking when a marked edge is found. Since we do this in the sorted order, all the edges between w and a are already marked. Thus, this marking process is done in total $O(n)$ time for all terminal pairs, and, by removing all unmarked edges, we can obtain the unique minimal Steiner forest F' of $(G, \mathcal{W})$ that contains F as a subgraph.

Theorem 25. Steiner Forest Enumeration *can be solved in $O(n + m)$ amortized time per solution and $O(n + m)$ space. If we use $O(nm)$ preprocessing time, this problem can be solved in $O(m)$ delay with $O(n^2)$ space.*

Proof. The correctness of the algorithm follows from Theorem 23.

Let X be a node in the enumeration tree. Let F be a partial Steiner forest of $(G, \mathcal{W})$ that is associated with X. We can find a terminal pair $W \in \mathcal{W}$ such that there are at least two valid paths for (F, W) in $O(n + m)$ time if it exists. In this case, there are at least two children of X, each of which can be generated in $O(n + m)$ delay with the s-t path enumeration algorithm. Otherwise, we can obtain the unique minimal Steiner forest F' of $(G, \mathcal{W})$ in $F + B$ in $O(n + m)$ time. Since each internal node of this "improved" enumeration tree has at least two children, the amortized running time of the algorithm is $O(n + m)$ using $O(n + m)$ space. By applying the output queue technique as in Theorem 15, we have the $O(n + m)$ delay bound with $O(n^2)$ space. □

5.1 Terminal Steiner trees

In this subsection, we give a linear-delay and linear-space enumeration algorithm for Terminal Steiner Tree Enumeration. The essential difference from Steiner Tree Enumeration is that every solution does not contain any terminals as an internal vertex, which can be easily handled. The analysis is almost the same as the one for Steiner Tree Enumeration. When $|W|$ is equal to two, the problem is identical to the s-t path enumeration problem. In this case, we can enumerate all minimal terminal Steiner tree in $O(n + m)$ delay and $O(n + m)$ space as proved in Theorem 12. Thus, in what follows, assume that $|W| > 2$.

Let $G = (V, E)$ be an undirected graph and let $W \subseteq V$ be a set of terminals. Recall that a Steiner tree T of (G, W) is called a terminal Steiner tree if every terminal in W is a leaf in T. A terminal Steiner tree T is called a *minimal terminal Steiner tree* of (G, W) if every proper subgraph of T is not a terminal Steiner tree of (G, W). It is straightforward to verify the following proposition.

Proposition 26. *T is a minimal terminal Steiner tree of (G, W) if and only if it is a terminal Steiner tree whose leaves are all terminal.*

We first characterize the condition that there is at least one terminal Steiner tree of (G, W).

Lemma 27. *Suppose that W has at least three terminals. Then, there is a terminal Steiner tree of (G, W) if and only if there is a component C in $G[V \setminus W]$ with $W \subseteq N_G(C)$. Moreover, every terminal Steiner tree of (G, W) has no edges between terminals and edges in a component C in $G[V \setminus W]$ with $W \setminus N_G(C) \neq \emptyset$.*

Proof. Clearly, if there is a terminal Steiner tree T of (G, W), then $G[V \setminus W]$ has a component C_T such that C_T contains T and $W \subseteq N_G(C_T)$ holds. For the converse, suppose that there is a component C in the statement. Then, we take a spanning tree T in C and add a leaf edge between w and a vertex in $V(T)$ for each $w \in W$. This can be done by the fact that $W \subseteq N_G(C)$, and hence the obtained graph is a terminal Steiner tree.

Suppose that there is a terminal Steiner tree T that has an edge e between terminals or in a component C of $G[V \setminus W]$ with $W \setminus N_G(C) \neq \emptyset$. Since there is a terminal w that is neither an end vertex of e nor contained in $N_G(C)$, the path between one of the end vertex of e and w must pass through some terminal $w' \neq w$, contradicting to the fact that T is a terminal Steiner tree. □

By Lemma 27, we assume that W is an independent set of G and every component C in $G[V \setminus W]$ satisfies $W \subseteq N_G(C)$.

Now, we define partial solutions and valid paths for TERMINAL STEINER TREE ENUMERATION. We say that a tree T in G is a *partial solution*, or more specifically, a *partial terminal Steiner tree* of (G, W) if either (1) T is the empty graph or (2) every leaf of T is terminal and there is a connected component C_T of $G[V \setminus W]$ such that $W \subseteq N(C_T)$ and $T \subseteq E(G[C_T \cup W])$. Suppose that T is a partial terminal Steiner tree that is not a terminal Steiner tree of (G, W). Let w be a terminal in $W \setminus V(T)$. Then, a *valid path P for (T, w)* is defined as: (1) a w-w' path in G for fixed $w' \in W \setminus \{w\}$ if T is the empty graph, or (2) a $(V(T) \setminus W)$-w path in $G[C_T \cup W]$. By the assumption that W is an independent set and every component C in $G[V \setminus W]$ satisfies $W \subseteq N_G(C)$, $T + E(P)$ is a partial terminal Steiner tree of (G, W). The following lemma is essential to show that TERMINAL STEINER TREE ENUMERATION can be solved by our strategy.

Lemma 28. *If T is a partial terminal Steiner tree of (G, W), then there is a minimal terminal Steiner tree of (G, W) that contains T as a subgraph.*

Proof. From the definition of a partial terminal Steiner tree, either T is the empty graph or there is a component C_T of $G[V \setminus W]$ such that $W \subseteq N_G(C)$ and $E(T) \subseteq E(G[C_T \cup W])$. Since T has no cycles, there is a spanning tree of C_T that contains all the edges in $E(T) \cap E(C_T)$. As $W \subseteq N_G(C_T)$, this spanning tree can be extended to a terminal Steiner tree of (G, W) by adding an edge $\{v, w\}$ for each $w \in W \setminus V(T)$ with some leaf $v \in N(w) \cap C_T$ and an edge $\{v, w\} \in T$ for $w \in W \cap V(T)$. This terminal Steiner tree may have a non-terminal leaf. By repeatedly removing such a non-terminal leaf and, by Proposition 26, we have a minimal terminal Steiner tree T^* of (G, W). Since every leaf of T is terminal, every edge in T is not removed in this process. Hence, T^* is a minimal terminal Steiner tree containing T. □

Now, we briefly describe an enumeration algorithm for terminal Steiner trees according to Algorithm 3. The main idea is similar to the algorithm for STEINER TREE ENUMERATION. We initially call E-Sol$(G, W, T_\emptyset)$, where $T_\emptyset$ is the empty graph. Let T be a partial terminal Steiner tree of (G, W). If T contains all the terminals in W, then T is a minimal terminal Steiner tree of (G, W) and we output it. Otherwise, there is a terminal w not contained in T. By the assumption that W is an independent set of G and every component C in $G[V \setminus W]$ satisfies $W \subseteq N_G(C)$, there is at least one valid path for (T, w). We extend T by adding a valid path P for (T, w). To do this, we need to enumerate all valid paths for (T, w). For case (1) where T is the empty graph, we just enumerate all w-w' paths in G. For case (2), we first compute the component C_T in $G[V \setminus W]$ and select a terminal w from $W \setminus V(T)$. Then, we enumerate all the valid paths for (T, w) by enumerating all $(V(T) \setminus W)$-w paths in $G[C_T \cup W]$. The correctness of the algorithm follows from an analogous argument in Lemma 14, and its complexity follows from an analysis almost identical to the one in Theorem 23. Thus, we have the following.

Theorem 29. TERMINAL STEINER TREE ENUMERATION *can be solved in $O(nm)$ delay and $O(n + m)$ space.*

The remaining of this subsection is devoted to showing a linear-delay algorithm for TERMINAL STEINER TREE ENUMERATION. To this end, we improve the enumeration tree so that each internal node has at least two children.

Lemma 30. *Let T be a non-empty partial Steiner tree of (G, W), let w be a terminal not contained in T, and let P be a valid path for (T, w). Then, P is the unique valid path for (T, w) in G if and only if every edge in P is a bridge in $G[C_t \cup W]$.*

Proof. Suppose that there are at least two valid paths P and P' for (T, w) in G. By an analogous argument in Lemma 16, there is a cycle in $G[C_t \cup W]$ that contains at least one edge from both paths P and P'. This implies that every $(V(T) \setminus W)$-w path in $G[C_T \cup W]$ has a non-bridge edge in $G[C_T \cup W]$.

Conversely, if P has a non-bridge edge e in $G[C_T \cup W]$, rerouting P along with a cycle passing through e yields a valid path distinct from P for (T, w). □

From this lemma, we can find either a terminal w such that $G[C_T \cup W]$ has at least two $V(T)$-w paths or a unique minimal terminal Steiner tree of (G, W) that contains T as a subgraph in linear time when T is a non-empty partial Steiner tree of (G, W). To do this, a similar idea used in STEINER TREE ENUMERATION works well. First, we enumerate all the bridges in $G[C_T \cup W]$ in linear time using a linear-time bridge enumeration algorithm [32]. Then, we compute an arbitrary minimal terminal Steiner tree T' of (G, W) that contains T as a subgraph in linear time. This can be done by taking an arbitrary spanning tree that contains T in $G[C_t \cup W]$ and removing non-terminal leaves. By Proposition 26, T' is a minimal terminal Steiner tree of (G, W). Using the bridges in $G[C_T \cup W]$ and T', by Lemma 30, we can "improve" the enumeration tree in the sense that each internal node has at least two children except for the root node. When T is the empty graph, we cannot apply Lemma 30. However, this exceptional case can be seen as a "linear-time preprocessing". Hence, by a similar argument to Theorem 25, we can obtain the following theorem.

THEOREM 31. *TERMINAL STEINER TREE ENUMERATION can be solved in $O(n + m)$ amortized time per solution and $O(n + m)$ space. If we allow $O(n^2)$ space and $O(nm)$ preprocessing time, this problem can be solved in $O(n + m)$ delay.*

5.2 Directed Steiner trees

In this subsection, we develop a linear-delay enumeration algorithm for minimal directed Steiner trees. Let $D = (V, E)$ be a directed graph with terminal set $W \subseteq V$ and let $r \in V \setminus W$. Recall that a subgraph T of D is a directed Steiner tree of (D, W, r) if T is a directed tree rooted at r that contains an r-w path for each $w \in W$ and T is a minimal directed Steiner tree of (D, W, r) if no proper subgraph of T is a directed Steiner tree of (D, W, r).

PROPOSITION 32. *T is a minimal directed Steiner tree of (D, W, r) if and only if it is a directed Steiner tree whose leaves are all terminal.*

Without loss of generality, for any vertex $v \in V$, D has a r-v directed path. By a similar argument as in the other variants, we say a directed tree T rooted at r is a *partial solution*, or more specifically, a *partial directed Steiner tree* of (D, W, r) if all the leaves in T are terminals. Let w be an arbitrary terminal not in a partial directed Steiner tree T. We say that a directed path P is *valid* for (T, w) if P is a directed $V(T)$-w path. Then, we observe the following key lemma.

LEMMA 33. *If T is a partial directed Steiner tree of (D, W, r), then there is a minimal directed Steiner tree of (D, W, r) that contains T as a subgraph.*

PROOF. Let T be a partial directed Steiner tree of (D, W, r). Then, from the assumption that each vertex is reachable from r, we can obtain a spanning tree T' containing T by adding paths to vertices not in T so that each vertex in T' can be reachable from r. Clearly, T' is a directed Steiner tree of (D, W, r) with root r. Repeatedly removing non-terminal leaves from T' yields a minimal directed Steiner tree of (D, W, r) by Proposition 32. □

We now describe our enumeration algorithm for minimal directed Steiner trees. We initially call $\texttt{E-Sol}(D, W, T_0 = (\{r\}, \emptyset))$. Let T be a partial directed Steiner tree of (D, W, r). On recursive call $\texttt{E-Sol}(D, W, T)$, if T contains all terminals, we output it. Otherwise, we find a terminal $w \in W \setminus V(T)$. Then, we compute all the valid paths for (T, w) by enumerating $V(T)$-w paths in D. By the assumption that w is reachable from r, there is at least one valid path for (T, w). For each valid path P for (T, w), we call $\texttt{E-Sol}(D, W, T + E(P))$. The next theorem directly follows from an analogous argument in Theorem 23.

THEOREM 34. *We can enumerate all minimal directed Steiner trees in $O(nm)$ delay and $O(n + m)$ space.*

In the remaining of this subsection, we give a linear-delay enumeration algorithm for DIRECTED STEINER TREE ENUMERATION. To this end, we "improve" the enumeration tree so that each internal node has at least two children. Once we can make this improvement in linear time for each node in the enumeration tree, we can prove, as in the previous subsections, that the entire algorithm runs in $O(n + m)$ amortized time and $O(n + m)$ space. To find either a terminal $w \in W \setminus V(T)$ such that there are at least two valid paths for

(T, w) or the unique minimal directed Steiner tree of (D, W, r) that contains T as a subgraph, we consider the multigraph $D' = D/E(T)$, which is obtained from D by contracting all edges in T. If D' has at least two directed r_T-w paths for some terminal $w \in W \setminus V(T)$, then we can immediately conclude that there are at least two valid paths for (T, w). To see this, let us consider an arbitrary depth-first search (DFS) tree T' in D' starting at node r_T corresponding to the contracted part T in D' and a total order $\prec$ on $V(T')$ determined by a post-order transversal in T'. Let $W' = W \setminus V(T)$ and let T^* be the minimal directed Steiner tree of (D', W', r_T) that is a subtree of T'.

LEMMA 35. *D' has a minimal directed Steiner tree of (D', W', r_T) distinct from T^* if and only if there exists a pair u and v of distinct vertices with $u \prec v$ in T^* such that $D' - E(T^*)$ has a directed v-u path.*

PROOF. For a directed path P, we denote by $t(P)$ the unique sink of P.

Suppose first that $D - E(T^*)$ has a directed v-u path P such that $u \prec v$. As $\prec$ is determined by a post-order transversal in T', either v is an ancestor of u in T', or they have no the ancestor-descendant relationship. If v is an ancestor of u in T', then $T^* - E(P') + E(P)$ is a minimal directed Steiner tree of (D', W', r_T) with distinct from T^*, where P' is defined to be the maximal subpath in T^* with $t(P') = u$ that has neither v nor any terminals as an internal vertex. Otherwise, let w be the lowest common ancestor of u and v in T^*. Then, $T^* - E(P') + E(P)$ is a minimal directed Steiner tree of (D', W', r_T) with distinct from T^*, where P' is defined to be the maximal subpath in T^* with $t(P') = u$ that has neither w nor any terminals as an internal vertex.

Conversely, let T^{**} be a minimal directed Steiner tree of (D', W', r_t) with $T^{**} \neq T^*$. Then, there exist two distinct r-w paths P^* and P^{**} in T^* and T^{**} for some $w \in W'$, respectively. Let Q be the maximal subpath of P^* with $V(Q) \subseteq V(P^*) \cap V(P^{**})$ and u be the source vertex of Q. Since $w \in V(P^*) \cap V(P^{**})$, u is well-defined. Let Q' be the maximal subpath of P^{**} with $t(Q') = u$ such that every internal vertex does not belong to T^* and let v be the source vertex of Q'. Since $r \in V(T^*) \cap V(P^{**})$, v is well-defined. As $P^* \neq P^{**}$, we have $u \neq v$.

Now we claim that $u \prec v$ in T^* and $D' - E(T^*)$ has a directed v-u path. If $v \prec u$, then either u is an ancestor of v in T^* or there is no directed path from v to u in D', which contradicts to the choice of u and v. Moreover, Q' is indeed a directed v-u path in $D' - E(T^*)$, completing the proof of the lemma. □

By Lemma 35, to find a terminal w that has at least two valid paths for (T, w), it is sufficient to check whether there is a directed path from a larger vertex to a smaller vertex in $D - E(T^*)$ with respect to $\prec$. This can be done as follows. From the largest vertex v with respect to $\prec$, we compute the reachability of each vertex in D' by a standard graph search algorithm. If there is a vertex u that is reachable from v, we are done. Otherwise, we remove all the vertices reachable from v and repeat the same procedure until we find such a path or the graph is empty. Since u is reachable from some v' with $u \prec v'$ in the original graph D' if and only if either u is reachable from v in D' or u is reachable from v' in the removed graph, this algorithm works correctly. Clearly, the algorithm runs in $O(n + m)$ time. Hence, we have the following conclusion.

THEOREM 36. *DIRECTED STEINER TREE ENUMERATION can be solved in amortized $O(n + m)$ time and $O(n + m)$ space. If we allow $O(nm)$ preprocessing time, this problem can be solved in $O(n + m)$ delay with $O(n^2)$ space.*

6 HARDNESS OF INTERNAL STEINER TREES AND GROUP STEINER TREES

In this section, we discuss some of the hard variants of STEINER TREE ENUMERATION. Recall that a Steiner tree of (G, W) is called an *internal Steiner tree* if every vertex in W is an internal vertex of the tree. Let $s, t \in V$ be distinct vertices and $W = V \setminus \{s, t\}$. Then, there is an internal Steiner tree of (G, W) if and only if G has an s-t Hamiltonian path, which implies the following theorem.

THEOREM 37. *Unless $P = NP$, there is no incremental-polynomial time algorithm for INTERNAL STEINER TREE ENUMERATION.*

We also remark that, as opposed to STEINER TREE ENUMERATION and several variants discussed above, we show that GROUP STEINER TREE ENUMERATION is at least as hard as MINIMAL TRANSVERSAL ENUMERATION. Given a hypergraph $\mathcal{H} = (U, \mathcal{E})$, MINIMAL TRANSVERSAL ENUMERATION is the problem of enumerating inclusion-wise minimal subsets $X \subseteq U$, called a *minimal transversal* of $\mathcal{H}$, such that $X \cap e \neq \emptyset$ for every $e \in \mathcal{E}$. The problem is one of the most challenging problems in the field of enumeration algorithms, and the best-known algorithm is due to Fredman and Khachiyan [13], which runs in total time $(|U| + N)^{O(\log(|U|+N))}$, where N is the number of minimal transversals of $\mathcal{H}$. The existence of an *output-polynomial time* algorithm, that is, it runs in total time $(U + |N|)^{O(1)}$, still remains open. Several papers [19, 27] pointed out some relation between the *minimum* group Steiner tree problem and the *minimum* transversal problem to prove the hardness of approximation. This relation also holds in the context of enumeration.

THEOREM 38. *If there is an algorithm that solves GROUP STEINER TREE ENUMERATION in output-polynomial time, then MINIMAL TRANSVERSAL ENUMERATION can be solved in output-polynomial time.*

PROOF. Let $\mathcal{H} = (U, \mathcal{E})$ be an instance of MINIMAL TRANSVERSAL ENUMERATION. Then, we construct a star graph G as follows. The center of G is denoted by r and G has a leaf vertex ℓ_u for each $u \in U$. For each $e \in \mathcal{E}$, we let $W_e = \{\ell_u : u \in e\}$ be a terminal set for e. It is not hard to see that X is a minimal transversal of $\mathcal{H}$ if and only if $G[X \cup \{r\}]$ is a minimal solution of $(G, \{W_e : e \in \mathcal{E}\})$ for GROUP STEINER TREE ENUMERATION. This indicates that GROUP STEINER TREE ENUMERATION is at least as hard as MINIMAL TRANSVERSAL ENUMERATION. □

7 MINIMAL INDUCED STEINER SUBGRAPHS FOR CLAW-FREE GRAPHS

Another variant of STEINER TREE ENUMERATION is INDUCED STEINER TREE ENUMERATION, where the goal is to enumerate all inclusion-wise minimal subsets of vertices that induce Steiner subgraphs of given (G, W). Since every induced subgraph is defined as a subset of vertices, we may not distinguish them unless confusion arises. Recall that a graph is *claw-free* if it has no induced subgraph isomorphic to $K_{1,3}$, i.e., a star with three leaves. Before describing the details of our proposed algorithm, we first observe that INDUCED

STEINER TREE ENUMERATION on claw-free graphs is a generalization of STEINER TREE ENUMERATION.

Let $G = (V, E)$ be a graph and let $W \subseteq V$ be terminals. We begin with the line graph $L(G)$ of G with $V(L(G)) = \{v_e : e \in E\}$ and two vertices v_e and v_e are adjacent in $L(G)$ if and only if they have a common end vertex, and then construct a graph H by adding vertices and edges to $L(G)$ as follows. Starting from $H = L(G)$, we add a vertex w' to H for each $w \in W$, and add an edge between v_e and w' for each $e \in \Gamma_G(w)$ in H. Define $W_H = \{v' : v \in W\}$. Then, the following theorem holds.

THEOREM 39. *Let T be a connected subgraph of G and let $V_T = \{v_e : e \in E(T)\}$. Then, T is a connected Steiner subgraph of (G, W) if and only if $H[V_T \cup W_H]$ is a connected induced Steiner subgraph of (H, W_H).*

PROOF. Suppose that $H[V_T \cup W_H]$ is a connected induced Steiner subgraph of (H, W_H). Observe that $H[V_T]$ is connected in H. To see this, suppose that $H[V_T]$ is not connected. Since $H[V_T \cup W_H]$ is connected, there is a terminal $w \in W_H$ and components X and Y in $H[V_T]$ such that both $N_H(w) \cap X$ and $N_H(w)$ are nonempty. However, $N_H(w)$ induces a clique in H for every $w \in W_H$, which is a contradiction. Thus, $H[V_T]$ is connected. Since V_T contains at least one vertex in $N_H(w)$ for each $w \in W$ and $H[V_T]$ is connected, T is a connected Steiner subgraph of (G, W).

Conversely, suppose that T is a connected Steiner subgraph of (G, W). For every pair of terminals w and w' in W, the edges of a path between them also induce a path between w and w' in W_H. This implies that $H[V_T \cup W_H]$ is a connected induced Steiner subgraph of (H, W_H). □

Since every line graph is claw-free, we conclude that INDUCED STEINER TREE ENUMERATION on claw-free graphs is a generalization of STEINER TREE ENUMERATION.

Our proposed algorithm is based on the *supergraph technique* [6, 7, 9, 23, 31]. Let us briefly describe the idea of this technique. Let $G = (V, E)$ be a claw-free graph and $W \subseteq V$. Let $\mathcal{S} \subseteq 2^V$ be the set of minimal induced Steiner subgraphs of (G, W). In this technique, we consider a directed graph $\mathcal{G}$ whose nodes correspond to the solutions $\mathcal{S}$, and whose arc set is defined so that $\mathcal{G}$ is strongly connected. As $\mathcal{G}$ is strongly connected, we can enumerate all the solutions from an arbitrary one by traversing $\mathcal{G}$. However, since strong connectivity is a "global" property and we do not know the entire node set $\mathcal{S}$ of $\mathcal{G}$, it would be nontrivial to "locally" define the set of arcs of $\mathcal{G}$, that is, define the neighborhood of each solution in $\mathcal{G}$. To this end, we define a "distance" measure $\sigma : \mathcal{S} \times \mathcal{S} \to \mathbb{Z}_{\geq 0}$ between two solutions. If one can prove that

(1) for $X, Y \in \mathcal{S}$, $\sigma(X, Y) = 0$ if and only if $X = Y$ and
(2) for every pair of distinct solutions X and Y, X has a neighbor Z in $\mathcal{G}$ with $\sigma(X, Y) > \sigma(Z, Y)$,

then $\mathcal{G}$ has a directed path from X to Y, and hence $\mathcal{G}$ is strongly connected. Specifically, for $X, Y \in \mathcal{S}$, we define $\sigma(X, Y) = |X \setminus Y|$. As we will see later, the supergraph $\mathcal{G}$ defined by a simple construction does not satisfy condition (1). To solve this issue, the second condition is relaxed to

(2') for every pair of distinct solutions X and Y, there is a directed path from X to Z in $\mathcal{G}$ with $\sigma(X, Y) > \sigma(Z, Y)$,

which is sufficient to prove the strong connectivity of $\mathcal{G}$. We can easily see the following proposition from the definition of σ and the minimality of solutions.

PROPOSITION 40. *Let X and Y be minimal induced Steiner subgraphs of (G, W). Then, $\sigma(X, Y) = 0$ if and only if $X = Y$.*

To complete the description of our enumeration algorithm, we need to define the neighborhood relation in $\mathcal{G}$. Given a connected vertex set $X \subseteq V$ with $W \subseteq X$, μ is a procedure that computes an arbitrary minimal induced Steiner subgraph $\mu(X, W)$ of (G, W) that is contained in X. Such a procedure is defined by a simple greedy algorithm that repeatedly removes a vertex v from $X \setminus W$ as long as $G[X \setminus \{v\}]$ is an induced Steiner subgraph of (G, W). To define a neighbor of X in $\mathcal{G}$, we need some observations. For each $v \in X \setminus W$, $G[X \setminus \{v\}]$ has two or more connected components as otherwise $G[X \setminus \{v\}]$ is an induced Steiner subgraph of (G, W), which contradicts the minimality of X. Since G is claw-free, there are exactly two connected components C_1 and C_2 in $G[X \setminus \{v\}]$. This follows from the fact that if $G[X \setminus \{v\}]$ has three components, v and three neighbors from these three components induce a claw $K_{1,3}$. As X is a minimal induced Steiner subgraph of (G, W), both components contain at least one terminal.

For each $w \in N(C_1) \setminus \{v\}$, we let $C_1^w = \mu(C_1 \cup \{w\}, (W \cap C_1) \cup \{w\})$ and $C_2^w = \mu(C_2, W \cap C_2)$. Note that C_1^w is well-defined as $G[C_1 \cup \{w\}]$ is connected. Let P be an arbitrary shortest path between w and C_2^w that avoids $N(C_1^w) \setminus \{w\}$. If such P exists, $C_1^w \cup C_2^w \cup V(P)$ is an induced Steiner subgraph of (G, W) since $C_1^w \cup C_2^w$ contains all the terminals and P is an $N(C_1^w)$-$N(C_2^w)$ path. Now, a neighbor Z of X is defined to be $\mu(C_1^w \cup C_2^w \cup V(P), W)$. As all the vertices in $V(P)$ are cut vertices in $G[C_1^w \cup C_2^w \cup V(P)]$, we have $V(P) \subseteq Z$. By the construction, Z does not contain v but does contain w. Given this, we say that Z is the *neighbor of X with respect to (v, w)* and define the arc set of $\mathcal{G}$ with this neighborhood relation. We only define such a neighbor Z when it is well-defined. Clearly, every minimal induced Steiner subgraph of (G, W) has $O(n^2)$ neighbors, which can be enumerated in $O(n^2(n + m))$ time.

Next, we show that for every pair of solutions X and Y, there is a solution Z reachable from X that is "closer to Y" than X. The following lemma directly implies the strong connectivity of $\mathcal{G}$.

LEMMA 41. *Let X and Y be distinct minimal induced Steiner subgraphs of (G, W). Then, $\mathcal{G}$ has a directed path from X to a minimal induced Steiner subgraph Z of (G, W) with $\sigma(X, Y) > \sigma(Z, Y)$.*

PROOF. Let v be a non-terminal vertex in $X \setminus Y$, and C_1 and C_2 be the connected components in $G[X \setminus \{v\}]$. Since both components contain at least one terminal each, there is a path $P = (w_1, \ldots, w_k)$ between $N(C_1)$ and $N(C_2)$ such that $V(P) \subseteq Y$. We can assume that $w_1 \in N(C_1) \setminus \{v\}$, $w_k \in N(C_2) \setminus \{v\}$, and all the other vertices are not in $N(C_1)$ by appropriately choosing P. Let $C_1^1 = \mu(C_1 \cup \{w_1\}, (W \cap C_1) \cup \{w_1\})$ and $C_2^1 = \mu(C_2, W \cap C_2)$. Since subpath $(w_2, \ldots, w_k)$ has no vertices in $N(C_1^1) \setminus \{w_1\}$, there is at least one shortest path P_1 between w_1 and $N(C_2^1)$ that avoids $N(C_1^1) \setminus \{w_1\}$. Define $X_1 = \mu(C_1^1 \cup C_2^1 \cup V(P_1), W)$ is the neighbor of X with

respect to (v, w_1). If $P_1 = P$, then

$$\sigma(X_1, Y) = |X_1 \setminus Y|$$
$$\leq |(C_1^1 \cup C_2^1 \cup V(P_1)) \setminus Y|$$
$$\leq |(C_1 \cup C_2) \setminus Y|$$
$$< |X \setminus Y|,$$

and hence we are done.

Suppose otherwise, that is, $P_1 \neq P$. Recall that $V(P_1) \subseteq X_1$. P_1 contains $\{w_1, \ldots, w_{i-1}\}$ and does not contain w_i for some $1 < i \leq k$. Let v' be the vertex adjacent to w_{i-1} with $v \neq w_{i-2}$ in P. Let C_1' and C_2' be the connected components in $G[X_1 \setminus \{v'\}]$. Define $C_1^2 = \mu(C_1' \cup \{w_i\}, (W \cap C_1') \cup \{w_i\})$ and $C_2^2 = \mu(C_2', W \cap C_2')$. We show that the following claim.

Claim: $C_2^2 \subseteq C_2$.

Proof of Claim: Let $P_1 = (w_1, \ldots, w_{i-1}, u_1, \ldots, u_t)$ with $u_1 = v'$ and $u_t \in N(C_2^1)$. Observe that $C_2' = C_2^1 \cup \{u_2, \ldots u_t\}$. In the following, we prove that the vertices $u_2, \ldots, u_t$ vanish using the function μ regardless of its implementation, which proves the claim as $C_2^1 \subseteq C_2$.

Since P_1 is a shortest path between w_1 and $N(C_2^1)$, every vertex in $\{u_2, \ldots, u_{t-1}\}$ is not adjacent to a vertex in C_2^1. This implies that those vertices are not contained in $C_2^2 = \mu(C_2', W \cap C_2')$. Suppose that there is a minimal Steiner subgraph C_2^2 of $(G[C_2'], W \cap C_2')$ containing u_t. We assume that C_2' contains at least two terminals as otherwise C_2^2 consists of exactly one vertex, which is terminal. Since u_t is in C_2^2, C_2^2 contains an induced path between two terminals in C_2^2 passing through u_t. Then, the two adjacent vertices of u_t in this path together with u_t and u_{t-1} form an induced claw, which contradicts the fact that G is claw-free. ☐

By the same argument as above, the neighbor X_2 of X_1 with respect to (v', w_i) is well-defined. We inductively compute the neighbor X_i of X_{i-1} unless P_i contains a vertex that does not belong to $V(P)$. Eventually, we have X_i that consists of $C_1^i \cup C_2^i \cup V(P_i)$, where $C_1^i \subseteq C_1 \cup V(P)$, $C_2^i \subseteq C_2$, and $V(P_i) \subseteq V(P)$. Then, we have

$$\sigma(X_i, Y) = |X_i \setminus Y|$$
$$\leq |(C_1 \cup C_2 \cup V(P)) \setminus Y|$$
$$\leq |(C_1 \cup C_2) \setminus Y|$$
$$< |X \setminus Y|,$$

which completes the proof of lemma. ☐

By Lemma 41, $\mathcal{G}$ is strongly connected. Thus, by using a standard graph search algorithm on $\mathcal{G}$, we can enumerate all the minimal induced Steiner subgraphs of (G, W) in $O(n^2(n + m))$ delay.

THEOREM 42. INDUCED STEINER SUBGRAPH ENUMERATION *can be solved in $O(n^2(n + m))$ delay using exponential space on claw-free graphs.*

REFERENCES

[1] Lowell W. Beineke. Characterizations of derived graphs. *J. Comb. Theory*, 9(2):129–135, 1970.

[2] Stephan Beyer and Markus Chimani. Strong steiner tree approximations in practice. *ACM J. Exp. Algorithmics*, 24(1), January 2019.

[3] 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 *Proc. SODA 2013*, pages 1884–1896, 2013.

[4] Marcus Brazil. *Steiner Minimum Trees in Uniform Orientation Metrics*, pages 1–27. Springer US, Boston, MA, 2001.

[5] Jarosław Byrka, Fabrizio Grandoni, Thomas Rothvoss, and Laura Sanità. Steiner Tree Approximation via Iterative Randomized Rounding. *J. ACM*, 60(1), 2013.

[6] Yixin Cao. Enumerating Maximal Induced Subgraphs. *CoRR*, abs/2004.09885, 2020.

[7] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.*, 74(7):1147–1159, 2008.

[8] Alessio Conte, Roberto Grossi, Mamadou Moustapha Kanté, Andrea Marino, Takeaki Uno, and Kunihiro Wasa. Listing induced steiner subgraphs as a compact way to discover steiner trees in graphs. In *Proc. MFCS 2019*, volume 138 of *LIPIcs*, pages 73:1–73:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[9] Alessio Conte and Takeaki Uno. New polynomial delay bounds for maximal subgraph enumeration by proximity search. In *Proc. of STOC 2019*, pages 1179–1190, 2019.

[10] Mitre Costa Dourado, Rodolfo Alves de Oliveira, and Fábio Protti. Algorithmic aspects of steiner convexity and enumeration of steiner trees. *Annals OR*, 223(1):155–171, 2014.

[11] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[12] David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998.

[13] Michael L. Fredman and Leonid Khachiyan. On the Complexity of Dualization of Monotone Disjunctive Normal Forms. *J. Algorithms*, 21(3):618–628, 1996.

[14] Michel X. Goemans, Neil Olver, Thomas Rothvoß, and Rico Zenklusen. Matroids and integrality gaps for hypergraphic steiner tree relaxations. In *Proc. STOC 2012*, pages 1161–1176. ACM, 2012.

[15] Martin Grötschel, Alexander Martin, and Robert Weismantel. The steiner tree packing problem in VLSI design. *Math. Program.*, 77:265–281, 1997.

[16] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. on Comput.*, 13(2):338–355, 1984.

[17] M. Hauptmann and M. Karpinski. A Compendium on Steiner Tree Problems. http://theory.cs.uni-bonn.de/info5/steinerkompendium/netcompendium.html. Accessed: 2021-06-19.

[18] John Hershberger, Matthew Maxel, and Subhash Suri. Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Trans. Algorithms*, 3(4):45–es, 2007.

[19] Edmund Ihler. The complexity of approximating the class steiner tree problem. In *Proc. WG 91*, volume 570 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 1991.

[20] Yoichi Iwata and Takuto Shigemura. Separator-based pruned dynamic programming for steiner tree. In *Proc. AAAI 2019*, pages 1520–1527. AAAI Press, 2019.

[21] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. on Comput.*, 4(1):77–84, 1975.

[22] Richard M. Karp. Reducibility among combinatorial problems. In *Proc. CCC 1972*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[23] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. Generating Cut Conjunctions in Graphs and Related Problems. *Algorithmica*, 51(3):239–263, 2008.

[24] Leonid G. Khachiyan, Endre Boros, Khaled M. Elbassioni, Vladimir Gurvich, and Kazuhisa Makino. On the complexity of some enumeration problems for matroids. *SIAM J. Discret. Math.*, 19(4):966–984, 2005.

[25] Benny Kimelfeld and Yehoshua Sagiv. Finding and approximating top-k answers in keyword proximity search. In *Proc. PODS 2006*, pages 173–182. ACM, 2006.

[26] Benny Kimelfeld and Yehoshua Sagiv. Efficiently enumerating results of keyword search over data graphs. *Inf. Syst.*, 33(4-5):335–359, 2008.

[27] Philip N. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *J. Algorithms*, 19(1):104–115, 1995.

[28] Guo-Hui Lin and Guoliang Xue. On the terminal steiner tree problem. *Inf. Process. Lett.*, 84(2):103–107, 2002.

[29] R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

[30] Yuya Sasaki. Cost-constrained minimal steiner tree enumeration by binary decision diagram. *CoRR*, abs/2104.06696, 2021.

[31] Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discret. Appl. Math.*, 117(1-3):253–265, 2002.

[32] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[33] T. Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical report, National Institute of Informatics Technical Report E, 2003.

[34] N.J. van der Zijpp and S. Fiorenzo Catalano. Path enumeration by finding the constrained K-shortest paths. *TRANSPORT RES B-METH*, 39(6):545 – 563, 2005.

[35] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Manag. Sci.*, 17(11):712–716, 1971.