Two Variations of the Minimum Steiner Problem[§]

Tsan-sheng Hsu[†], *Kuo-Hui Tsai*[*‡], *Da-Wei Wang*[†‡¶] and *D. T. Lee*[†⋆]

August 12, 2003

Abstract

Given a set S of starting vertices and a set T of terminating vertices in a graph $G = (V, E)$ with non-negative weights on edges, the *minimum Steiner network* problem is to find a subgraph of G with the minimum total edge weight. In such a subgraph, we require that for each vertex $s \in S$ and $t \in T$, there is a path from s to a terminating vertex as well as a path from a starting vertex to t. This problem can easily be proven NP-hard. For solving the minimum Steiner network problem, we first present an algorithm that runs in time and space that both are polynomial in n with constant degrees, but exponential in $|S| + |T|$, where n is the number of vertices in G. Then we present an algorithm that uses space that is quadratic in n and runs in time that is polynomial in n with a degree $O(\max\{\max\{|S|, |T|\} - 2, \min\{|S|, |T|\} - 1\})$. In spite of this degree, we prove that the number of Steiner vertices in our solution can be as large as $|S| + |T| - 2$. Our algorithm can enumerate all possible optimal solutions. The input graph G can either be undirected or directed acyclic. We also give a linear time algorithm for the special case when $\min\{|S|, |T|\} = 1$ and $\max\{|S|, |T|\} = 2$.

The *minimum union paths* problem is similar to the minimum Steiner network problem except that we are given a set H of hitting vertices in G in addition to the sets of starting and terminating vertices. We want to find a subgraph of G with the minimum total edge weight such that the conditions required by the minimum Steiner network problem are satisfied as well as the condition that every hitting vertex is on a path from a starting vertex to a terminating vertex. Furthermore, G must be directed acyclic. For solving the minimum union paths problem, we also present algorithms that have a time and space tradeoff similar to algorithms for the minimum Steiner network problem. We also give a linear time algorithm for the special case when $|S| = 1$, $|T| = 1$ and $|H| = 2$.

[§]An extended abstract of part of this paper appears in [7].

[†]Institute of Information Science, Academia Sinica, Nankang 11529, Taipei, Taiwan, ROC. E-mail: {tshsu, wdw, dtlee}@iis.sinica.edu.tw.

[‡]Supported in part by the National Science Council, Taiwan, ROC, under Grant No. NSC-83-0408-E-001-021.

[*]Department of Computer Science, National Taiwan Ocean University, Taiwan, ROC. Email: tsaikh@mail.ntou.edu.tw.

[¶]Corresponding author.

[⋆] Supported in part by the National Science Foundation under Grants CCR-9309743 and INT-9207212, and by the Office of Naval Research under Grant No. N00014-93-1-0272.

1 Introduction

Given a graph $G = (V, E)$, a non-negative cost on each edge in E, and a set of vertices $Z \subset V$, the *minimum Steiner problem* is to find a minimum cost subgraph with a given property which spans Z. The vertices besides Z in the subgraph are *Steiner vertices*. For example, the *minimum Steiner tree problem* is one in which the subgraph is a *tree*. The minimum Steiner problem has many important applications and has been extensively studied [2, 8, 9, 15, 16]. In one of the latest survey papers [8], more than 300 references are listed.

Let S and T be two subsets of vertices of V. A subgraph G' of G is said to be S,T-*connected* if for each vertex $s \in S$ there exists a path[1] in G' from s to a vertex $t \in T$, and for each vertex $t \in T$ there exists a path in G' from a vertex $s \in S$ to t. We consider the following two variations of the minimum Steiner problem on graph $G = (V, E)$ with a non-negative weight on each edge. The *minimum Steiner network* (MSN) problem is defined as follows. Given a set of starting vertices S and a set of terminating vertices T, we want to find a subgraph G^* with the minimum total edge weight such that G^* is S,T-connected if such a G^* exists. The *minimum union paths* (MUP) problem is a generalized version of the minimum Steiner network problem. An additional set of *hitting* vertices $H \subset V$ is specified, and the subgraph G^*, if exists, is required to contain H, and each vertex in H must be in a path from a starting vertex to a terminating vertex. Note that the minimum Steiner network problem is a special case of the minimum union paths problem in which the hitting set H is empty. It can be seen that the above two minimum Steiner problems are NP-hard by a polynomial transformation from the exact cover by the 3-sets problem (for an example, see the discussion in Chapter 1 of [9]).

Several important applications for solving the minimum Steiner problem on a directed acyclic graph are mentioned in [12] for finding a minimum Steiner arborescence. The two problems defined here can be used to solve generalized versions of these problems. For example, finding a minimum Steiner network can be used to design an optimal drainage system for a building where starting vertices are places to dump wastes and terminating vertices are sewers. Finding minimum union paths can be used to design an optimal system layout for an irrigation and drainage system where starting vertices are water sources, hitting vertices are places that need water, and terminating vertices are drains.

There are several types of algorithms that can be used to find an exact solution of the minimum Steiner problem which include dynamic programming [3], 0-1 linear programming [1], and exhaustive enumeration [10]. Though most of the discussions are on undirected

[1] In this paper, we refer to a *path* as a sequence of vertices $a_1, a_2, \ldots, a_k$ such that (a_i, a_{i+1}), $1 \leq i < k$, is a directed edge pointed from a_i to a_{i+1} and $a_i \neq a_j$ for all $i \neq j$, i.e., every path is a *simple path*.

graphs, they can be easily extended to directed graphs.

The time complexity of well-known approaches for solving the Steiner-tree-like problem is (1) exponential in the number of terminal vertices by a dynamic programming approach [3], or (2) polynomial in the number of vertices in the input graph with a degree equaling the number of Steiner vertices in a general graph by enumerating all possible candidates of Steiner vertices [10]. The dynamic programming approach uses more space than the enumerating approach. In this paper, we first develop a dynamic programming approach to solve our problems, then we develop our efficient enumerating algorithm.

A typical exhaustive enumeration algorithm for the minimum Steiner tree problem makes use of the fact that a minimum Steiner tree is a minimum spanning tree on the set of vertices in the solution. Thus, by properly choosing the set of Steiner vertices, we can easily find the solution by computing a minimum spanning tree. Therefore, finding an accurate upper bound on the number of Steiner vertices is crucial in estimating the running time of this type of algorithm. Let k be the above upper bound. The time complexity of the algorithm is $O(n^k \cdot MST(G))$, where $MST(G)$ denotes the time complexity of finding a minimum spanning tree of G. A similar algorithm can be devised to solve the MSN problem and the MUP problem in time exponential in k. We shall show that by exploiting some properties of the solutions to the two problems, we can solve them in time exponential roughly in $\frac{k}{2}$.

Some related work for variations of the minimum Steiner problem on directed graphs can be found in [6, 11, 12, 13, 17]. In particular, Nastansky *et al.* [12] presented an efficient heuristic algorithm for solving the minimum Steiner arborescence problem on a directed acyclic graph by exhaustive enumeration. However, they do not provide any theoretical analysis on the time complexity of their algorithm, which we believe is exponential in k. Rao *et al.* [13] described an approximation algorithm for finding a minimum Steiner arborescence in a rectilinear plane rooted at the origin with the additional constraint that all paths from the root to the leaves must be the shortest.

In this paper, let n and m be the number of vertices and edges in G, respectively. In the MSN problem, let $\alpha = \max\{|S|, |T|\}$, let $\beta = \min\{|S|, |T|\}$ and let $\sigma = |S| + |T|$. In the MUP problem, let $\gamma = |S| + |H| + |T|$.

Results in our paper are summarized in Table 1.

2 Minimum Steiner Network

For ease of description, we assume the graph G is directed acyclic. However, all of our discussions also hold for the case when G is undirected. We also omit the word "directed" when it does not cause confusion, and minimality of a solution network is assumed everywhere

		Polynomial-time algorithms	Quadratic-space algorithms	Special case
MSN	time	$O(nm + n{\cdot}3^{\sigma} + n^2{\cdot}2^{\sigma})$	$O(nm + 2^{\sigma}{\cdot}\alpha^3{\cdot}n^{\max\{\alpha-2,\beta-1\}})$	$\alpha = 2,\ \beta = 1$
				$O(n+m)$
	space	$O(n^3{\cdot}2^{\sigma})$	$O(n^2)$	$O(n+m)$
MUP	time	$O(nm + \gamma!{\cdot}n{\cdot}((18\gamma)^{\gamma} + n{\cdot}(8\gamma)^{\gamma}))$	$O(nm + \gamma!{\cdot}(8\gamma)^{\gamma}{\cdot}\gamma^3{\cdot}n^{\gamma-1})$	$\lvert S\rvert = \lvert T\rvert = 1,\ \lvert H\rvert = 2$
				$O(n+m)$
	space	$O(n^3{\cdot}4^{\gamma})$	$O(n^2)$	$O(n+m)$

Table 1: Summary of results.

in the discussion.

Given two non-empty disjoint subsets of vertices S and T in G, a *Steiner network* of G is a subgraph G^* of G such that G^* is S,T-connected. The vertices in S are called *starting vertices* and those in T are *terminating vertices*. Any path in a Steiner network starting from a vertex in S and ending in a vertex in T is called an *S-T path*. If the edges in G are associated with non-negative weights, then a Steiner network with a minimum total edge weight is called a *minimum Steiner network*[2] and denoted as $\mathrm{MSN}(G,S,T)$. Note that in a minimum Steiner network except for the two extreme vertices an *S-T path* may contain vertices in S or T.

2.1 Preliminaries

Before we describe our algorithm for finding a minimum Steiner network, we examine its properties. Through these properties, we give a structural description of a minimum Steiner network. Then we construct a minimum Steiner network by first efficiently enumerating all networks satisfying the above description and then picking one with the minimum cost.

Let $S\dot{\cup}T\dot{\cup}Z$ be the set of vertices in $\mathrm{MSN}(G,S,T)$, where $\dot{\cup}$ is the disjoint set union operator. The vertices in Z, which contain no vertices in S and T, are *Steiner vertices*. We define the *distance network* for G, denoted $D(G)$, to be the graph with the same set of vertices as G and there is an edge (u,v) in $D(G)$ if and only if there is a path between u and v. The edge (u,v) is given a weight equal to the total weight of a shortest path from u to v in G.

Lemma 2.1 *Given S and T, the cost of MSN(G,S,T) is equal to the cost of MSN$(D(G),S,T)$. Furthermore, given MSN$(D(G),S,T)$, MSN(G,S,T) can be constructed in $O(n+m)$ time.*

[2]Here we consider *the* Steiner network with the least number of edges among those with a minimum total edge weight.

Proof: Let $P(u,v)$ denote a maximal path in $\mathrm{MSN}(G,S,T)$ connecting vertices u and v for some $u,v \in V$ such that the intermediate nodes in the path have in-degrees and out-degrees equal to 1. This lemma simply states that if we replace every such path $P(u,v)$ in $\mathrm{MSN}(G,S,T)$ with an edge (u,v) whose weight is equal to the total edge weight of all the edges in $P(u,v)$, then the resulting network has a cost equal to $\mathrm{MSN}(D(G),S,T)$. Note that the set of Steiner vertices Z in $\mathrm{MSN}(G,S,T)$ can be partitioned into two sets $Z = Z_1 \dot{\cup} Z_2$, where Z_2 contains the intermediate vertices of all such $P(u,v)$ defined above and Z_1 will be the set of Steiner vertices in $\mathrm{MSN}(D(G),S,T)$. □

From Lemma 2.1, we know that it suffices to compute a minimum Steiner network from the distance network of the graph. We will focus our discussion on finding a minimum Steiner network in a distance network in order to show that in an acyclic graph, the possible candidates for Steiner vertices can be enumerated efficiently.

A Steiner vertex in $\mathrm{MSN}(D(G),S,T)$ is *convergent* if its in-degree is greater than 1, and is *divergent* if its out-degree is greater than 1. From Lemma 2.1, any Steiner vertex in a minimum Steiner network of a distance network must be either convergent or divergent. Note that a vertex may be both convergent and divergent if its in-degree and out-degree are both greater than 1.

Lemma 2.2 *Given a directed acyclic graph G with non-negative edge weights and two disjoint subsets of vertices S and T, let $s \in S$, $t \in T$, and P be a path from s to t in $MSN(D(G),S,T)$ whose length (number of edges) is greater than 1. Let $P = [s, w_1, w_2, \ldots, w_q, t]$, $q \geq 1$. Then there exists b, $0 \leq b \leq q$, such that exactly one of the following two conditions is true: (1) $w_i, 1 \leq i \leq b$, is convergent and $w_i, b \leq i \leq q$, is divergent; (2) $w_i, 1 \leq i \leq b$, is convergent and $w_i, b+1 \leq i \leq q$, is divergent.*

Proof: We prove this lemma by contradiction. Recall that $\mathrm{MSN}(D(G),S,T)$ has a minimal number of edges. Let b be the largest index such that $w_1, \ldots, w_{b-1}$, and w_b are convergent in P. Assume that $w_{b+1} \neq t$ is *strictly* divergent, i.e., not convergent, and that w_a is convergent, where $a > b+1$. Let H be the resulting graph obtained by removing the edge from w_{a-1} to w_a from P, and hence from $\mathrm{MSN}(D(G),S,T)$. Note that H remains a Steiner network since w_a is convergent and has another incoming edge. Since the weight of each edge is non-negative, either the cost of H is less than that of $\mathrm{MSN}(D(G),S,T)$ or H has a smaller number of edges. Thus $\mathrm{MSN}(D(G),S,T)$ is not optimal, which is a contradiction. □

The vertex w_b in the path P as specified in Lemma 2.2 is referred to as a *neutral* vertex. Note that if a vertex is neutral in an S-T path, then it is neutral in any other S-T path.

Corollary 2.3 *There exists at most one neutral vertex in the path P as specified in Lemma 2.2.* □

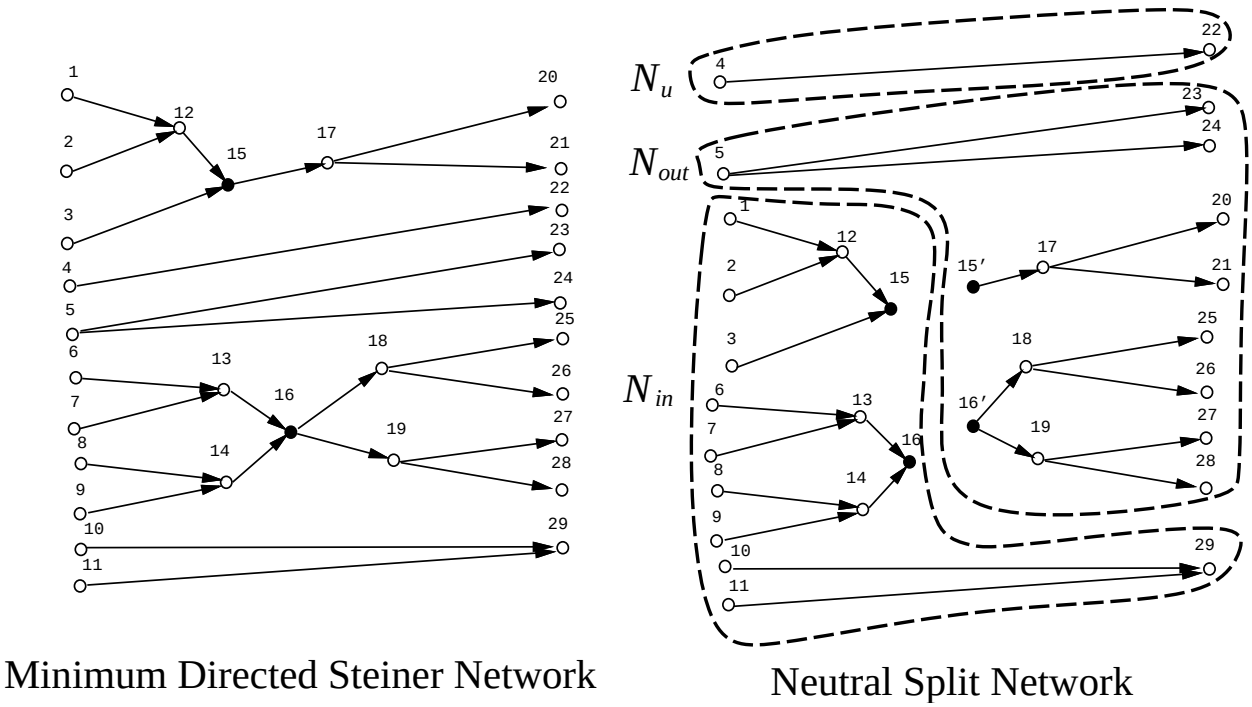

Figure 1: A minimum Steiner network, the set of neutral vertices shown in solid circles, and its neutral split network.

A directed graph is an *incoming arborescence* if either it is an isolated vertex or there exists exactly one *root vertex* u such that there is exactly one path from any other vertex to u. A collection of incoming arborescences is an *incoming forest*. A directed graph is an *outgoing arborescence* if it is an isolated vertex or there exists exactly one root vertex u such that there is exactly one path from u to any other vertex in the graph. A collection of outgoing arborescences is an *outgoing forest*. A vertex is a *leaf* in an incoming (respectively, outgoing) forest if its in-degree (respectively, out-degree) is zero. A vertex in a forest that is not a leaf is an *internal vertex*. Given a vertex u in a minimum Steiner network a *split graph at* u is the graph obtained by replacing u with two vertices u_1 and u_2, such that u_1 inherits all incoming edges of u, and u_2 inherits all outgoing edges of u. Given $\mathrm{MSN}(D(G), S, T)$, let its *neutral split network* be defined by applying the split operation on every neutral vertex in $\mathrm{MSN}(D(G), S, T)$. The cost of the neutral split network is the same as the cost of the original minimum Steiner network. Figure 1 illustrates a minimum Steiner network and its neutral split network. The set of neutral vertices are shown in solid circles, while the others are shown in hollow circles.

Lemma 2.4 *There are at most $\sigma - 2$ Steiner vertices in MSN$(D(G), S, T)$, and the bound is tight.*

Proof: We prove this lemma by induction on the value of σ. This lemma is obviously true when $|S| = 1$ and $|T| = 1$. For the induction step, observe that adding a vertex to S or T creates at most one Steiner vertex in $D(G)$, by converting a vertex from $\mathrm{MSN}(G, S, T)$ in Z_2 into one which is in Z_1. To prove that the bound is tight, let us construct, as described below, a graph G such that $\mathrm{MSN}(D(G), S, T)$ has exactly $\sigma - 2$ Steiner vertices. Let B_S be an incoming arborescence with all equal weight edges such that the root is u, the set of leaves is S and the in-degree of each node except for the leaves (each of whose in-degree is 0) is 2 and its out-degree is 1 except for the root (whose out-degree is 0). For convenience B_S is called an incoming binary tree. Let B_T be similarly defined except that it is an outgoing arborescence (binary tree) whose root is v, and the set of leaves is T. Then $G = B_S \cup B_T \cup \{(u, v)\}$ is a graph whose $\mathrm{MSN}(D(G), S, T)$ contains $\sigma - 2$ Steiner vertices. □

Lemma 2.5 *The neutral split network of a minimum Steiner network MSN$(D(G), S, T)$ for an acyclic graph G with non-negative edge weights can be partitioned into three disjoint subgraphs N_u, N_{in} and N_{out} where (1) N_u consists of edges from a vertex in S to a vertex in T; (2) N_{in} is an incoming forest where each leaf is a vertex in S, each non-root internal vertex is a Steiner vertex, and each root is either a Steiner vertex or a vertex in T; (3) N_{out} is an outgoing forest where each leaf is a vertex in T, each non-root internal vertex is a Steiner vertex, and each root is either a Steiner vertex or a vertex in S. There are no isolated vertices in N_{in} and N_{out}.*

Proof: Since G is acyclic, $\mathrm{MSN}(D(G), S, T)$ is also acyclic. Every vertex in $\mathrm{MSN}(D(G), S, T)$ is in an S-T path in $\mathrm{MSN}(G, S, T)$. Thus this lemma follows from Lemma 2.2 and Corollary 2.3. □

Corollary 2.6 *The number of arborescences in N_{in} (respectively, N_{out}) is no more than half of the number of leaves in N_{in} (respectively, N_{out}).*

Proof: There is no isolated vertex in N_{in} or N_{out}. In each arborescence, there are at least two leaves. Hence the corollary holds. □

Now we examine properties of the three disjoint subgraphs in a neutral split network. Note that N_{in} is a forest. We will exploit properties of an incoming arborescence, denoted as N'_{in}, constructed as follows from N_{in} Let u be a vertex not in the neutral split network of $\mathrm{MSN}(D(G), S, T)$. N'_{in} is constructed by adding to N_{in} a new vertex u and an edge from each root in N_{in} to u. All new edges have zero weight. Let $D(G)'$ be the resulting graph obtained by adding the same set of vertex and edges to $D(G)$.

Lemma 2.7 *The graph N'_{in} is a minimum Steiner incoming arborescence in $D(G)'$ and can be obtained in $O(nm + \ell^2 \cdot n^{\ell - t})$ time, where $\ell > 2$ is the number of leaves in N_{in} and t the number of trees in N_{in}.*

Proof: It takes $O(nm)$ time to find the distance network for a directed acyclic graph [14]. Note that the distance network contains $O(n^2)$ edges. The minimum spanning arborescence in a graph with x vertices and y edges can be found in $O(x \cdot \log x + y)$ time [5]. There are less than $\ell - t$ Steiner vertices and $O(\ell^2)$ edges in a minimum Steiner arborescence. The number of subgraphs of G with less than $\ell - t$ vertices is $O(n^{\ell-t})$. For each subgraph above, the time to find its minimum spanning arborescence is $O((\ell - t)\log(\ell - t) + \ell^2) = O(\ell^2)$. N'_{in} is a minimum spanning arborescence of a subgraph described above. Hence the lemma holds. □

Given N'_{in}, N_{in} can be obtained in time linear in the number of vertices in N_{in}. It is also true that N_{in} is a *minimum Steiner incoming arborescence* in $D(G)$. Similarly we can construct an outgoing arborescence N'_{out} by adding to N_{out} a new vertex u and and an edges from u to each root node in N_{out}. Let $D(G)''$ be the resulting graph obtained by adding the same set of edges to $D(G)$ as in the previous construction. The weight of each added edge is zero.

Lemma 2.8 *N'_{out} is a minimum Steiner outgoing arborescence in $D(G)''$ and can be obtained in $O(nm + \ell^2 \cdot n^{\ell-t})$ time, where $\ell > 2$ is the number of leaves in N_{out} and t the number of trees in N_{out}.* □

It can be easily proven that given N'_{out}, N_{out} can be obtained in time linear in the number of vertices in N'_{out}. It is also true that N_{out} is a *minimum Steiner outgoing arborescence* in $D(G)$.

Corollary 2.9 *Let ℓ_{in} be the number of leaves in N_{in} and let ℓ_{out} be the number of leaves in N_{out}.*
(1) The number of internal vertices in N_{in} is less than ℓ_{in}.
(2) The number of internal vertices in N_{out} is less than or equal to ℓ_{out}.

Proof: The degree of each internal vertex in N_{in} is at least 2. Thus (1) holds by using a simple induction argument. Note that the degree of the root in a tree in N_{out} could be one. Thus (2) holds by using a simple induction argument. □

2.2 Dynamic Programming Approach

We first give an algorithm that runs in time and space that are both polynomial in n with constant degrees, but are exponential in σ. Our approach is similar to the one used in [3].

Let $Y_1 \subseteq S$ and let $Y_2 \subseteq T$. We define two restricted versions of the minimum Steiner network problem as follows. Given a vertex $i \in Z \cup S$, let $\mathrm{MSN}_i^1(D(G), Y_1 \cup \{i\}, Y_2)$ be a

minimum Steiner network under the condition that i must be an internal convergent vertex. Given a vertex $i \in Z \cup T$, let $\text{MSN}_i^2(D(G), Y_1 \cup \{i\}, Y_2)$ be a minimum Steiner network under the condition that i must be an internal divergent vertex.

Let $W(Y_1, Y_2)$ be the total edge weight for $\text{MSN}(D(G), Y_1, Y_2)$. Let $W_i^1(Y_1, Y_2)$ be the total edge weight for $\text{MSN}_i^1(D(G), Y_1 \cup \{i\}, Y_2)$. Let $W_i^2(Y_1, Y_2)$ be the total edge weight for $\text{MSN}_i^2(D(G), Y_1, Y_2 \cup \{i\})$.

Let H be a solution for $\text{MSN}_i^1(D(G), Y_1 \cup \{i\}, Y_2)$. Thus the total edge weight of H is $W_i^1(Y_1, Y_2)$. The split graph of H at i contains two minimum spanning networks $\text{MSN}(D(G), Y_1', \{i\})$ and $\text{MSN}(D(G), Y_1 \cup \{i\} \setminus Y_1', Y_2)\}$, where Y_1' is the subset of vertices in the component that contains the vertex i. We can obtain a similar formulation for $W_i^2(Y_1, Y_2)$. We then have the following equations:

$$W_i^1(Y_1, Y_2) = \min_{\forall Y_1' \subset Y_1} \{W(Y_1', \{i\}) + W(Y_1 \cup \{i\} \setminus Y_1', Y_2)\}, \textit{and} \tag{1}$$

$$W_i^2(Y_1, Y_2) = \min_{\forall Y_2' \subset Y_2} \{W(Y_1, Y_2 \cup \{i\} \setminus Y_2') + W(\{i\}, Y_2')\}. \tag{2}$$

Let $a_{i,j}$ be the edge weight between vertices i and j in $D(G)$. To find $W(\{i\} \cup Y_1, Y_2)$, the optimal solution will be in one of the following four cases depending on the type of vertex. *Case 1.* i is internally convergent; $W_i^1(Y_1, Y_2)$ is the solution. *Case 2.* i is internally divergent; $W_i^2(Y_1, Y_2)$ is the solution. *Case 3.* i is a leaf and points to a vertex in Y_1; $\min_{\forall j \in Y_1}\{a_{i,j} + W(Y_1, Y_2)\}$ is the solution. *Case 4.* i is a leaf and points to an internal vertex which makes the vertex pointed to by i an internal convergent vertex; $\min_{\forall j \notin Y_1}\{a_{i,j} + W_j^1(Y_1, Y_2)\}$ is the solution. Based on the four cases discussed above, we have the following formula:

$$W(\{i\} \cup Y_1, Y_2) = \min \left\{ \begin{array}{l} W_i^1(Y_1, Y_2) \\ W_i^2(Y_1, Y_2) \\ \min_{\forall j \in Y_1}\{a_{i,j} + W(Y_1, Y_2)\} \\ \min_{\forall j \notin Y_1}\{a_{i,j} + W_j^1(Y_1, Y_2)\} \end{array} \right\}. \tag{3}$$

Similarly, we derive the following formula.

$$W(Y_1, \{i\} \cup Y_2) = \min \left\{ \begin{array}{l} W_i^1(Y_1, Y_2) \\ W_i^2(Y_1, Y_2) \\ \min_{\forall j \in Y_1}\{a_{i,j} + W(Y_1, Y_2)\} \\ \min_{\forall j \notin Y_1}\{a_{i,j} + W_j^2(Y_1, Y_2)\} \end{array} \right\}. \tag{4}$$

We list the algorithm for computing a minimum Steiner network below.

1. Computer $D(G)$;
2. For each $Y_1 \subseteq S$ and each $Y_2 \subseteq T$ do
   (a) For each vertex $i \in V$ do
      i. If $i \in Y_1$, then
      Compute and store $W_i^1(Y_1, Y_2)$ according to Equation 1;
      ii. If $i \in Y_2$, then
      Compute and store $W_i^2(Y_1, Y_2)$ according to Equation 2;
      iii. Compute $W(\{i\} \cup Y_1, Y_2)$ using Equation 3 and store the corresponding minimum Steiner network;
      iv. Compute $W(Y_1, \{i\} \cup Y_2)$ using Equation 4 and store the corresponding minimum Steiner network;
3. Return minimum Steiner network with the least stored $W(S, T)$ value.

Let $Y_1', Y_1'' \subseteq S$ and $Y_2', Y_2'' \subseteq T$. In implementing Step 2, if $|Y_1'| \geq |Y_1''|$ and $|Y_2'| \geq |Y_2''|$, then the loop body with indexes $Y_1 = Y_1''$ and $Y_2 = Y_2''$ is executed before the loop body with indexes $Y_1 = Y_1'$ and $Y_2 = Y_2'$. That is, the loop is executed in the order of $(|Y_1|, |Y_2|) = (0,0), (0,1), (1,0), (1,1), (1,2), (2,1), \ldots, (|S|, |T|)$. Note that for a fixed $(|Y_1|, |Y_2|)$, we execute all possible candidates for Y_1 and Y_2 in random order.

Given fixed i, Y_1 and Y_2, it takes $O(2^{|Y_1|})$ (respectively, $O(2^{|Y_2|})$) computations of the W function to find $W_i^1(Y_1, Y_2)$ (respectively, $W_i^2(Y_1, Y_2)$). In total, there are $O(n \cdot 2^\sigma)$ different values for W_i^1 and W_i^2. Steps 2(a)i and 2(a)ii must compute the W function $(n \cdot 3^\sigma)$ times.

There are $O(n \cdot 2^\sigma)$ formulas of the form Equations 3 or 4. Each of them involves a computation of a minimum of $O(n)$ terms. Hence it takes $O(n^2 \cdot 2^\sigma)$ time to compute all W functions. Overall, Steps 2(a)iii and 2(a)iv take $O(n \cdot 3^\sigma + n^2 \cdot 2^\sigma)$ time and $O(n \cdot 2^\sigma)$ space. Note that we need $O(nm)$ time and $O(n^2)$ space to compute and store a minimum Steiner network of a directed acyclic graph [14]. Hence we have the following theorem:

Theorem 2.10 *We can find MSN$(D(G), S, T)$ in $O(nm + n \cdot 3^\sigma + n^2 \cdot 2^\sigma)$ time and $O(n^3 2^\sigma)$ space.* □

2.3 Enumerating Approach

We now give an algorithm that uses $O(n^2)$ space, but runs in time that is polynomial in n with a degree $O(\max\{\alpha - 2, \beta - 1\})$.

Using Lemmas 2.5, 2.7, and 2.8, we can construct a minimum Steiner network as follows. First we obtain (by exhaustive enumeration) $S_1 \subseteq S$ and $T_1 \subseteq T$ such that $|S_1| = |T_1|$, and construct N_u by finding a minimum-cost bipartite perfect matching between S_1 and T_1 in $D(G)$. We then obtain the set of roots $\mathcal{R}$ (by exhaustive enumeration) in the neutral split

network of MSN$(D(G), S, T)$. Note that $\mathcal{R}$ can be partitioned into roots which are neither in S nor in T, roots which are in $S \setminus S_1$, and roots which are in $T \setminus T_1$. That is, $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3$ where $\mathcal{R}_1 \subseteq V \setminus (S \cup T)$, $\mathcal{R}_2 \subseteq (S \setminus S_1)$, and $\mathcal{R}_3 \subseteq (T \setminus T_1)$. After choosing $\mathcal{R}$, we find a minimum Steiner incoming forest N_{in} on the set of leaves $S \setminus (S_1 \cup \mathcal{R})$ and the set of roots $\mathcal{R} \setminus \mathcal{R}_2$. By Corollary 2.9, there can be at most $|S| - |S_1| - 1$ Steiner vertices (including vertices in $\mathcal{R}$). We also find a minimum Steiner outgoing forest N_{out} on the set of leaves $T \setminus (T_1 \cup \mathcal{R})$ and the set of roots $\mathcal{R} \setminus \mathcal{R}_3$. By Corollary 2.9, there can be at most $|T| - |T_1|$ Steiner vertices (including vertices in $\mathcal{R}$). Note that the sets of Steiner vertices in N_{in} and in N_{out} are disjoint. Because the way S_1, T_1, and $\mathcal{R}$ are chosen, the cost of MSN$(D(G), S, T)$ is equal to the cost of $N_u \cup N_{in} \cup N_{out}$. MSN$(D(G), S, T)$ is easily constructed from N_u, N_{in}, and N_{out}. The algorithm for the above discussion is as follows.

1. Compute $D(G)$;
2. For each $S_1 \subseteq S$, $T_1 \subseteq T$ and $|S_1| = |T_1| = i$ do
   (a) Compute and store N_u formed by a minimum-cost bipartite matching between S_1 and T_1 in $D(G)$;
   (b) For each $\mathcal{R} \subseteq V$ and $j = |\mathcal{R}| \leq \alpha - i$ do
      i. Let $\mathcal{R}_2 = \mathcal{R} \cap (S \setminus S_1)$, let $\mathcal{R}_3 = \mathcal{R} \cap (T \setminus T_1)$ and let $\mathcal{R}_1 = \mathcal{R} \setminus (\mathcal{R}_1 \cup \mathcal{R}_2)$;
      ii. For each $W \subseteq V$ and $|W| \leq |S| - i - j - 1$ do
         - Compute a minimum Steiner incoming forest on the set of leaves $S \setminus (S_1 \cup \mathcal{R}$, the set of internal vertices W and the set of roots $\mathcal{R}_1 \cup \mathcal{R}_3$;

      Let N_{in} be a forest with the least cost among all forests found above;
      iii. For each $W \subseteq V$ and $|W| \leq |T| - i - j$ do
         - Compute a minimum Steiner outgoing forest on the set of leaves $S \setminus (T_1 \cup \mathcal{R}$, the set of internal vertices W and the set of roots $\mathcal{R}_1 \cup \mathcal{R}_2$;

      Let N_{out} be a forest with the least cost among all forests found above;
   (c) Make $N_{in} \cup N_{out} \cup N_u$ a candidate for MSN$(D(G), S, T)$;
3. Return a solution with the least total edge weight among all candidates.

From the above discussion, we have the following theorem:

Theorem 2.11 *MSN(G, S, T) can be constructed in $O(nm + 2^{\sigma} \cdot \alpha^3 \cdot (n^{\alpha-2} + n^{\beta-1}))$ time and $O(n^2)$ space.*

Proof: Without loss of generality assume that $\alpha = |S|$ and $\beta = |T|$. Note that $|S_1| = |T_1| \leq \beta$. Our algorithm first constructs the distance network (Step 1). We then enumerate all candidates for a directed minimum Steiner network in the distance network. After

each candidate has been enumerated, we record its total weight. After all candidates have been enumerated, any one that has the minimum total weight is a solution we want. By Lemma 2.5, a candidate solution can be partitioned into the three disjoint networks N_u, N_{in} and N_{out}. Hence we construct all possible candidate solutions by using all possible choices for N_u, N_{in} and N_{out}. By using properties proven in the previous section, we can greatly reduce the number of choices for N_u, N_{in} and N_{out}.

Let i be the number of pairs of vertices in N_u. Note that $i \leq \beta$. Hence there are up to $\binom{|S|}{i}\binom{|T|}{i}$ possible different choices for the set of vertices in N_u.

Recall the definition of $\mathcal{R}$ in the neutral split network of $\mathrm{MSN}(D(G), S, T)$. Let $|\mathcal{R}| = j$. By Corollary 2.6, $j \leq \left\lfloor \frac{\alpha-i}{2} \right\rfloor + \left\lfloor \frac{\beta-i}{2} \right\rfloor$, which is no more than $\alpha - i$. There are up to $\binom{n}{j}$ candidates for the set of vertices $\mathcal{R}$. Once we pick $\mathcal{R}$, we partition $\mathcal{R}$ into $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ as specified in Step 2(b)i. Note that the set of vertices in N_{in} is $\mathcal{R}_1 \cup \mathcal{R}_3 \cup (S \setminus \mathcal{R}_2)$ and the set of vertices in N_{out} is $\mathcal{R}_1 \cup \mathcal{R}_2 \cup (T \setminus \mathcal{R}_3)$. We enumerate choices for N_{in} and N_{out} by enumerating the choices for $\mathcal{R}$ and then finding minimum Steiner forests as specified in Steps 2(b)ii and 2(b)iii.

Thus our algorithm takes time

$$O((A) + \sum_{i=1}^{\beta} \binom{|S|}{i}\binom{|T|}{i} \sum_{j=1}^{\alpha-i} \binom{n}{j} [(B) + (C) + (D)]),$$

where (A) is the time needed to construct the distance network (Step 1) from G and is equal to $O(nm)$ for a directed acyclic graph [14]; (B) is the time needed to find a minimum cost bipartite perfect matching in a bipartite graph (Step 2a) with $2{\cdot}i$ vertices and $O(i^2)$ edges and is $O((2{\cdot}i)^2 \log(2{\cdot}i) + (2{\cdot}i){\cdot}i^2)$ [4]; (C) is the time needed to find a minimum Steiner incoming forest with j roots, $|S| - i$ leaves, and up to $|S| - i - j - 1$ internal vertices given $D(G)$ (Step 2(b)ii); and (D) is the time needed to find a minimum Steiner outgoing forest with j roots, $|T| - i$ leaves, and up to $|T| - i - j$ internal vertices given $D(G)$ (Step 2(b)iii). By Lemmas 2.7 and 2.8, we know that (C) is $O(|S|^2{\cdot}n^{|S|-i-j-1})$ and (D) is $O(|T|^2{\cdot}n^{|T|-i-j})$. Thus (B)+(C)+(D) $= O(\alpha^2{\cdot}(n^{\alpha-i-j-1} + n^{\beta-i-j}))$.

$$\text{Since} \quad \binom{n}{j} \leq n^j, \binom{n}{j}((B) + (C) + (D)) = O(\alpha^2{\cdot}(n^{\alpha-i-1} + n^{\beta-i})).$$

$$\text{Thus} \quad \sum_{j=1}^{\alpha-i} \binom{n}{j}((B) + (C) + (D))) = O(\alpha^3{\cdot}(n^{\alpha-i-1} + n^{\beta-i})).$$

$$\text{Since} \quad \sum_{i=1}^{\beta} \binom{|S|}{i}\binom{|T|}{i} \leq \sum_{i=1}^{\beta} \binom{\beta}{i} \sum_{i=1}^{\alpha} \binom{\alpha}{i} \text{ and } \sum_{i=1}^{\alpha} \binom{\alpha}{i} = 2^\alpha - 1,$$

$$\sum_{i=1}^{\beta} \left(\begin{array}{c} |S| \\ i \end{array} \right) \left(\begin{array}{c} |T| \\ i \end{array} \right) = O(2^{\alpha} \cdot 2^{\beta}).$$

Thus, the overall time complexity is $O(nm + 2^{\sigma} \cdot \alpha^3 \cdot (n^{\alpha-2} + n^{\beta-1}))$. It takes $O(n^2)$ space to store the distance network. Hence the theorem holds. □

By Lemma 2.4, the number of Steiner vertices in a minimum Steiner network can be as large as $\sigma - 2$. Finding a minimum Steiner network by brute force may require doing a minimum spanning arborescence computation $O(n^{|\sigma-2})$ times. From Theorem 2.11 it appears that a minimum Steiner network can be found by performing a minimum spanning arborescence computation $O(2^{\sigma} \cdot \alpha\} \cdot (n^{\alpha-2} + n^{\beta-1}))$ times. This substantially saves computation time if α is $O(\log n)$.

Remark: Within the same time complexity, we can solve the *strong minimum Steiner network* problem in which no intermediate vertices in any S-T path can belong to S and T. In other words, all starting vertices are of in-degree 0 (sources) and all terminating vertices are of out-degree 0 (sinks). This problem can be solved by finding MSN(G', S, T) where G' is obtained by removing all incoming edges to a starting vertex and all outgoing edges to a terminating vertex.

Within the same time complexity, we can also solve the MSN problem on an undirected graph, in which each S-T path in the solution is undirected, by applying the algorithm in Sections 2.2 and 2.3. The correctness of the algorithm, which is established in properties proved in Section 2.1 for the directed case, can be applied to our algorithm for the undirected case as well. Observe that a solution for an undirected minimum Steiner network contains no cycles. Thus there is a unique orientation for the undirected solution by orienting edges in every S-T path from a vertex in S towards a vertex in T. All properties in Section 2.1 can be applied to this oriented solution.

2.4 A Linear Time Algorithm When $\alpha = 2$ and $\beta = 1$

Note that if $\beta = 1$, then finding a minimum Steiner network is equivalent to finding a minimum Steiner arborescence. Let the (i, j)-MSN problem denote the MSN problem with i starting vertices and j terminating vertices. In Figure 2, we list the four possible configurations for the (1,2)-MSN problem.

For the solving the MSN problem in which $\alpha = 2$, the following result shows that the computation of the distance network can be avoided when $|S \cup T| = 3$.

Theorem 2.12 *If $\beta = 1$ and $\alpha = 2$, then MSN(G, S, T) can be computed in $O(n + m)$ (instead of $O(n \cdot m)$) time and $O(n + m)$ (instead of $O(n^2)$) space.*

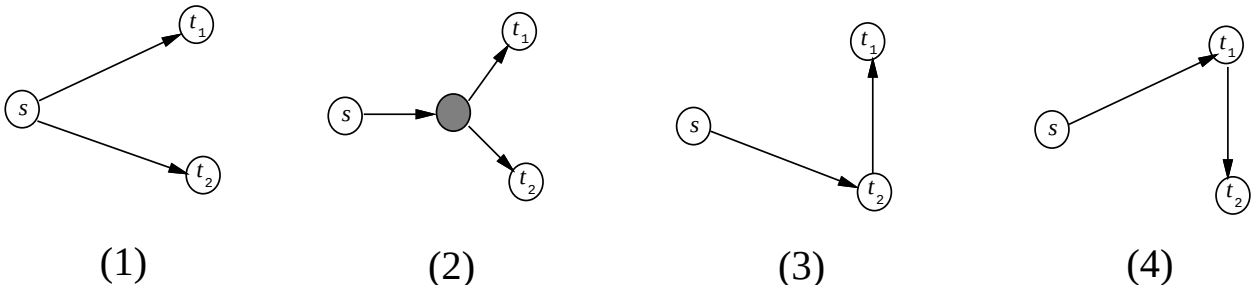

Figure 2: All possible configurations for a solution of the (1,2)-MSN problem in the distance network where the starting vertex is s and the set of terminating vertices is $\{t_1, t_2\}$. The shaded vertex is a Steiner vertex.

Proof: Note that there is at most one Steiner vertex in MSN$(D(G), S, T)$. Any path from any vertex u to another vertex v in MSN(G, S, T) is also a shortest path from u to v in G. Without loss of generality assume that $|S| = 1$ and $|T| = 2$. Our algorithm first computes the single source shortest paths (in G) for the vertex in S. Let $r(G)$ be the resulting graph obtained by reversing the direction of each edge in G. We then compute the single source shortest paths (in $r(G)$) for each of the two vertices in T. The above computation takes $O(m)$ time for a directed acyclic graph.
We distinguish two cases.

Case 1: There is no Steiner vertex.

The cost of a minimum Steiner network is equal to the sum of edge-costs in the two paths from the vertex in S to the two vertices in T (Figure 2.(1)), or in the *one* path from the vertex in S to the two vertices in T (Figures 2.(3) and 2.(4)).

Case 2: There is exactly one Steiner vertex u.

Note that u could be a vertex in T and that this case reduces to those shown in Figures 2.(3) and 2.(4). We can try all possible candidates for u. Whenever we fix a candidate for u, the cost of the Steiner network with only one Steiner vertex u can be computed in constant time by adding the shortest path distance (in G) from the vertex in S to u, and the costs of the two shortest paths (in $r(G)$) from the two vertices in T to u.

Hence the theorem holds. □

Remark: We can also solve in linear time and space the MSN problem on an undirected graph in which each S-T path in the solution is undirected by using the same algorithm.

3 The Minimum Union Paths Problem

Given a weighted directed acyclic graph $G = (V, E)$ and three non-empty mutually disjoint subsets of vertices S, H, and T, a set of *S-H-T union paths* of G is a subgraph G' of G with the following properties.

1. For every $s \in S$ there is a path in G' from s to a vertex in T.
2. For every $t \in T$ there is a path in G' from a vertex in S to t.
3. For every $h \in H$ there is a path in G' passing through h which starts from some vertex in S and ends with some vertex in T.

The set of vertices S is referred to as *starting vertices*, the set of vertices H as *hitting vertices* and the set of vertices T as *terminating vertices*. The *minimum union paths* problem is to find a set of *S-H-T minimal* union paths in G with a minimum total edge weight.[3] A set of minimum union paths that has the minimum total edge weight is $\mathrm{MUP}(G, S, H, T)$.

Lemma 3.1 *There are at most $\gamma + |H| - 4$ Steiner vertices in $MUP(D(G), S, H, T)$ for $|H| \geq 1$.*

Proof: We prove this lemma by induction on $\gamma + |H|$. Note that $\gamma + |H| = |S| + |T| + 2 \cdot |H|$. This lemma is trivially true when $|S| = 1$, $|T| = 1$ and $|H| = 1$. For the induction step, we observe that adding a vertex to S or T creates at most one Steiner vertex and that adding a vertex to H creates at most two Steiner vertices. Hence the lemma is true. □

Similar to Lemma 2.1 we have the following.

Lemma 3.2 *$MUP(G, S, H, T)$ can be constructed from $MUP(D(G), S, H, T)$ in linear time and space.* □

If $|S| > 1$ or $|T| > 1$, then we augment the given graph by adding to G two new vertices s_0 and t_0 and the set of edges $E^* = \{(s_0, s) \mid s \in S\} \cup \{(t, t_0) \mid t \in T\}$. The edge weight of each added edge is an arbitrary positive constant. Let $\tilde{G}$ be the augmented graph. From $\mathrm{MUP}(\tilde{G}, \{s_0\}, S \cup H \cup T, \{t_0\})$ we can obtain the solution $\mathrm{MUP}(G, S, H, T)$ by deleting the vertices s_0 and t_0 and all their incident edges. Henceforth, without loss of generality we may assume that $|S| = |T| = 1$ and we also consider the distance network $D(G)$ of G.

[3] Here we also consider the network with the smallest number of edges among those that have a minimum total edge weight.

3.1 Preliminaries

Before we present a solution to MUP$(D(G), \{s\}, H, \{t\})$, we first discuss properties of an optimal solution. Given MUP$(D(G), \{s\}, H, \{t\})$, note that s has no incoming edge and t has no outgoing edge. Let L_i denote the subset of vertices in H such that for every vertex $v \in L_i$ there is a path from s to v containing at most i vertices in H. Note that when we traverse any path from s to t in MUP$(D(G), \{s\}, H, \{t\})$, the first vertex in H encountered is in L_1. Let ℓ be the smallest integer such that $L_\ell = L_{\ell+1}$. Then MUP$(D(G), \{s\}, H, \{t\})$ has ℓ levels.

According to the definition of L_i, we partition H as follows. Let $H_1 = L_1$ and $H_i = L_i \setminus L_{i-1}$ for all $2 \leq i \leq \ell$. Let $H_0 = \{s\}$ and let $H_{\ell+1} = \{t\}$.

Lemma 3.3 *It is not possible to have a path in MUP$(D(G), \{s\}, H, \{t\})$ from a vertex in H_i to a vertex in H_j, if $i + 1 < j$ and $j \neq \ell + 1$.* □

A vertex $v \in H_i$ is *backward* if there is a path $P^*_{v,v'}$ from v to a vertex $v' \in H_j$, $j \leq i$, that does not pass through any other vertex in H. The vertex v' is a *backward successor* of v. $P^*_{v,v'}$ is called a *backward path.* A vertex $v \in H_i$ is *forward* if there is a path $P^*_{v,t}$ from v to t that does not pass through any other vertex in H. $P^*_{v,t}$ is called a *forward path.* The following lemma states that a vertex cannot be both forward and backward. Furthermore, there is at most one forward or backward path.

Lemma 3.4 *For any vertex in H there exists at most one forward or one backward path, but not both.*

Proof: If a vertex v is both forward and backward, let P_f and P_b be its forward and backward paths, respectively. Let $e = (x, y)$ be the first edge, not contained in P_b, encountered when we traverse P_f starting from v. Since t has no outgoing edge, e must exist. Let G' be the graph resulting from removing edge e from MUP$(D(G), \{s\}, H, \{t\})$. For a $\{s\}$-$\{t\}$ path P in MUP(D(G),{s},H,{t}) that passes through e, we can find an $\{s\}$-$\{t\}$ path P' in G' by unioning: (1) sub-path of P from s to x, (2) sub-path of P_b from x to the backward successor u of v, and (3) any path from u to t. Thus G' is a set of $\{s\}$-H-$\{t\}$ union paths with a smaller total edge weight or a smaller number of edges than MUP$(D(G), \{s\}, H, \{t\})$. This is a contradiction. Thus we know v cannot be both forward and backward.

We now suppose that v has two backward paths P_1 and P_2. Let e be the first edge, not contained in P_2, encountered when we traverse P_1 starting from v. Then by an argument similar to the one given in the last paragraph, we can derive a contradiction. Similarly one can prove that it is impossible for v to have more than one forward path. □

An example of a set of minimum union paths is shown in Figure 3.

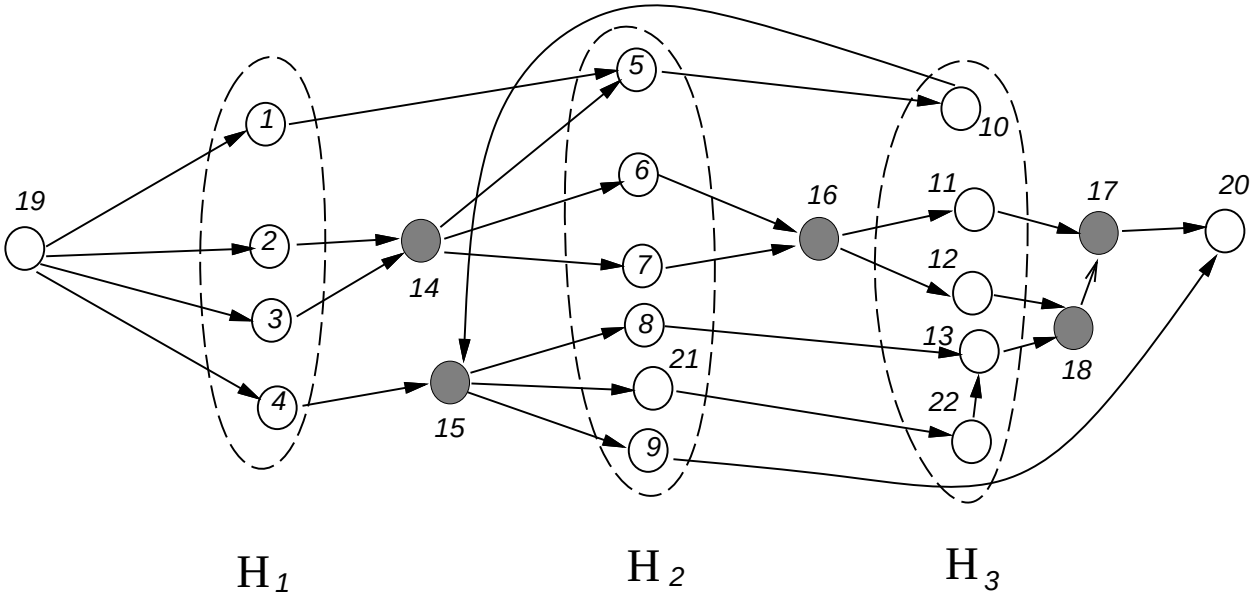

Figure 3: A set of minimum union paths on the starting vertex set {19}, the hitting vertex set {1,2,3,4,5,6,7,8,9,10,11,12,13,21,22} and the terminating vertex set {20}. The hitting set is partitioned into $H_1 = \{1, 2, 3, 4\}$, $H_2 = \{5, 6, 7, 8, 9, 21\}$ and $H_3 = \{10, 11, 12, 13, 22\}$. Vertex 10 is a backward vertex whose backward index is 1. Vertex 22 is also a backward vertex whose backward index is 2. Vertex 9 is a forward vertex.

Let H_f (respectively, H_b) be the set of forward (respectively, backward) vertices in H. Let H'_i be the set of vertices in H_i that is neither forward nor backward. We define a *split extension graph* for $\mathrm{MUP}(D(G), \{s\}, H, \{t\})$ as follows. For every vertex $u \in (H_f \cup H_b)$, we split u into two vertices u_{in} and u_{out} where u_{in} inherits all incoming edges and u_{out} inherits all outgoing edges. The rest of the edges and the vertices remain unchanged.

Given a backward vertex u whose backward successor is in H_i, its *backward index* $bi(u)$ is $i - 1$. The backward index of a forward vertex is ℓ and the backward index of a vertex that is neither forward nor forward is ∞. The *level index* of u, $\ell i(u) = i$ if $u \in H_i$. We partition edges in the split extension graph for $\mathrm{MUP}(D(G), \{s\}, H, \{t\})$ into $G_0 \cup G_1 \cup \cdots \cup G_\ell$ where G_i is the induced subgraph of $\mathrm{MUP}(D(G), \{s\}, H, \{t\})$ on the two sets of vertices $\overline{H}_i = H'_i \cup \{u_{out} \mid bi(u) = i\}$ and $\underline{H}_i = H'_{i+1} \cup \{u_{in} \mid \ell i(u) = i + 1\}$. The sets of vertices $\overline{H}_0, \overline{H}_1, \ldots, \overline{H}_\ell$, and $\underline{H}_0, \underline{H}_1, \ldots, \underline{H}_\ell$ are the *pairwise extension* of $H_0, H_1, \ldots,$ and $H_{\ell+1}$.

In Figure 4, we illustrate an example of the pairwise extension of the set of minimum union paths in Figure 3.

Lemma 3.5 *G_i is a minimum Steiner network MSN$(G, \overline{H}_i, \underline{H}_i)$.*

Proof: Note that G_i is a directed Steiner network in G for the set of starting vertices $\overline{H}_i$ and the set of terminating vertices $\underline{H}_i$. If G_i is not a minimum Steiner network, then let G'_i

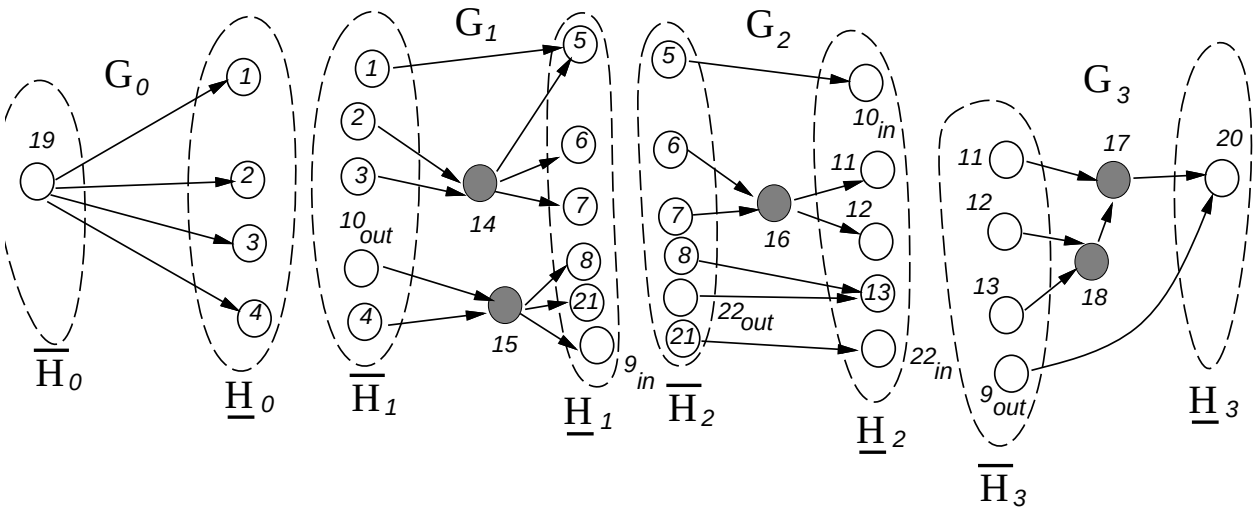

Figure 4: The pairwise extension and the split partition graphs of the minimum union paths shown in Figure 3.

be a minimum Steiner network. We delete edges in G_i from MUP$(D(G), \{s\}, H, \{t\})$ and replace them with edges in G'_i. We obtain a set of union paths with either smaller total edge weights or a smaller number of edges. This is a contradiction. □

3.2 The Algorithms

Given MUP$(D(G), \{s\}, H, \{t\})$, we define its *configuration* to be the pairwise extension of $H_0, H_1, \ldots,$ and $H_{\ell+1}$. By Lemma 3.5, we can find an MUP$(D(G), \{s\}, H, \{t\})$ as follows. We first find its configuration, then we construct each G_i by using the algorithms in Sections 2.2 and 2.3 to find a minimum Steiner network. Our algorithm finds the correct configuration by enumerating all possible configurations given H.

Lemma 3.6 *There are $O(k!{\cdot}2^k{\cdot}k^k)$ possible configurations for MUP$(D(G), \{s\}, H, \{t\})$, where $k = |H|$.*

Proof: Let $\mathcal{T}(k)$ be the number of different ways that one can partition a set of k vertices. Then $\mathcal{T}(0) = 0$ and

$$\mathcal{T}(k) = \sum_{i=1}^{k} \left(\begin{array}{c} k \\ i \end{array} \right) \cdot \mathcal{T}(k-i).$$

Thus, $\mathcal{T}(k) = O(k!{\cdot}2^k)$. Given a partition for H, each vertex in H can be either a forward vertex, a backward vertex, or a vertex that is neither forward nor backward. The backward index of a backward vertex can also have $\ell - 1$ choices. However, the total number of choices is less than or equal to k. Thus there are $O(k^k)$ different pairwise extensions for a partition. Hence the lemma holds. □

Theorem 3.7 *MUP*$(D(G), \{s\}, H, \{t\})$ *can be found in either* $O(nm+k!\cdot n((18\cdot k)^k+n\cdot(8k)^k))$ *time and* $O(n^3\cdot 4^k)$ *space, or* $O(k!\cdot(8\cdot k)^k\cdot k^3\cdot n^{k-1}+nm)$ *time and* $O(n^2)$ *space, where* $k = |H|$.

Proof: Algorithm for implementing this theorem is as follows.

1. Compute $D(G)$;
2. /* Enumerate all possible configurations. */
   For each possible partition $(H_1, H_2, \ldots, H_\ell)$ of H do
   (a) For each possible combination of backward indices of vertex in H do
      i. construct the corresponding pairwise extension $\overline{H}_0, \overline{H}_1, \ldots, \overline{H}_\ell$, $\underline{H}_0, \underline{H}_1, \ldots,$ and $\underline{H}_\ell$;
      ii. For each $i \leq \ell$ do
         - Compute $G_i = \text{MSN}(D(G), \overline{H}_i, \underline{H}_i)$;
      iii. Make $G_0 \cup G_1 \cup \cdots \cup G_\ell$ a candidate for MUP$(D(G), \{s\}, H, \{t\})$;
3. Return a solution with the least total edge weight found among all candidates.

Note that $|\overline{H}_i| \leq k$ and $|\underline{H}_i| \leq k$ for all $0 \leq i \leq \ell$. Note also that $|\cup_{i=0}^{\ell}(\overline{H}_i \cup \underline{H}_i)| \leq 2k+2$. Given a partition extension of a partition of H, the time and space needed in Step 2(a)ii to compute all G_i is bound by computing (k,k)-MSN which are either $O(nm + n\cdot 3^{2k} + n^2\cdot 2^{2k})$ time and $O(n^3\cdot 2^{2k})$ space by Theorem 2.10, or $O(4^k\cdot k^3\cdot n^{k-1} + nm)$ time and $O(n^2)$ space by Theorem 2.11, where $O(nm)$ is the time to compute the distance network. Given G, the distance network for G needs only be computed once. The number of times that Step 2a being executed is analyzed in Lemma 3.6. Hence the theorem holds. □

Theorem 3.8 *MUP*$(D(G), S, H, T)$ *can be found in Either* $O(nm + \gamma!\cdot n((18\cdot\gamma)^\gamma + n(8\gamma)^\gamma))$ *time and* $O(n^3\cdot 4^\gamma)$ *space, or* $O(\gamma!\cdot(8\cdot\gamma)^\gamma\cdot\gamma^3\cdot n^{\gamma-1} + nm)$ *time and* $O(n^2)$ *space.*

Proof: By Lemma 3.2 and Theorem 3.7. □

By Lemma 3.1, the number of Steiner vertices in a minimum Steiner network can be as large as $\gamma + |H| - 4$. Finding a minimum Steiner network by brute force may require doing a minimum spanning arborescence computation $O(n^{\gamma+|H|-4})$ times. Theorem 3.8 states that by using our algorithm, a set of minimum union paths can be found by performing a minimum spanning arborescence computation $O(\gamma!\cdot(8\cdot\gamma)^\gamma\cdot\gamma\cdot n^{\gamma-1})$ times. For a fixed γ our algorithm runs in $O(nm+n^{\gamma-1})$ time, while a brute force approach takes $O(nm+n^{\gamma+|H|-4})$ time. Thus our algorithm runs asymptotically faster when $|H|$ is a constant greater than 3.

3.3 A Linear Time Algorithm When $|S| = |T| = 1$ and $|H| = 2$

Let the (i,k,j)-MUP problem denote the MUP problem with i starting vertices, k hitting vertices and j terminating vertices. In this section, we give a linear-time-and-space algorithm

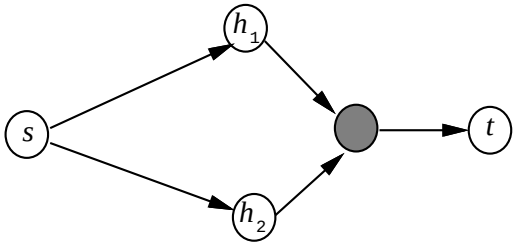

Figure 5: A possible solution for the (1,2,1)-MUP problem in the distance network where the starting vertex is s, the terminating vertex is t and the set of hitting vertices is $\{h_1, h_2\}$. The shaded vertex is a Steiner vertex.

to solve the (1,2,1)-MUP problem. Note that the original algorithm needs to compute a distance network and thus takes at least $O(nm)$ time. We will show that the computation of the distance network can be avoided as in Section 2.4 for the (1,2)-MSN problem.

We first analyze all possible configurations of a solution for the (1,2)-MSN problem when there is one starting vertex and two terminating vertices. As shown below, the solution for the (1,2,1)-MUP problem can be decomposed into the union of solutions to two instances of the (1,2)-MSN problem.

In Figure 2, we list the four possible configurations for the (1,2)-MSN problem. The solution for the (1,2,1)-MUP problem can be easily obtained by the union of solutions to two (1,2)-MSN problems. For example, the solution in Figure 5 is a union of type (1) in Figure 2 and the graph obtained from type (2) in Figure 2 after reversing the directions of edges. Note that if the first part of the solution is type (3) or (4) in Figure 2, we only have to find a path from a hitting vertex to the terminating vertex to form a desired solution.

Thus, to find the solution for the (1,2,1)-MUP problem with one starting vertex s, two hitting vertices h_1 and h_2 and one terminating vertex t, we first find $\mathrm{MSN}(G, \{s\}, \{h_1, h_2\})$. By Theorem 2.12, this can be done in linear time and space. According to the discussion in Section 3.1, the set H of hitting vertices can be partitioned into at most two levels.
We distinguish two cases.

Case 1: The hitting vertices are in the same level.

This occurs when $\mathrm{MSN}(G, \{s\}, \{h_1, h_2\})$ is either type (1) or (2) in Figure 2. Let $r(G)$ be the resulting graph obtained from G by reversing the direction of each edge in G. We find $\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\})$. Since both $\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\})$ and $\mathrm{MSN}(G, \{s\}, \{h_1, h_2\})$ are optimal solutions and G is acyclic, no vertex other than h_1 and h_2 can appear in both $\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\})$ and $\mathrm{MSN}(G, \{s\}, \{h_1, h_2\})$.

If $\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\})$ is either type (1) or (2) in Figure 2, then the solution

we want is $\mathrm{MSN}(G, \{s\}, \{h_1, h_2\}) \cup r(\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\}))$. If $\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\})$ is either type (3) or (4) in Figure 2, without loss of generality, let h_1 be the vertex with in-degree 0 in $r(\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\}))$. A shortest path from s to h_1 together with $r(\mathrm{MSN}(r(G), \{t\}, \{h_1, h_2\}))$ is the desired solution.

Case 2: The two hitting vertices are in different levels.

This occurs when $\mathrm{MSN}(G, \{s\}, \{h_1, h_2\})$ is either type (3) or (4) in Figure 2. Without loss of generality, let h_1 be the hitting vertex whose out-degree is 0 in $\mathrm{MSN}(G, \{s\}, \{h_1, h_2\})$. We complete the computation by finding a shortest path from h_1 to t.

Hence we have the following theorem.

Theorem 3.9 *The (1,2,1)-*MUP *problem can be solved in linear time and space.* □

Remark: When $|H|$ is small, enumerating all possible partitions of H is an effective way to find a solution for the MUP problem.

4 Concluding Remarks

We have described an enumerative approach to solve two variations of the minimum-cost Steiner problem on a directed acyclic graph with non-negative edge weights. Properties of the solutions were presented and used to obtain algorithms more efficiently than brute force enumerating methods for these two problems. Whether or not additional properties of the graph can be used to improve the time complexity of the algorithms that compute optimal solutions remains to be seen.

References

[1] Y. P. Aneja. An integer linear programming approach to the Steiner problem in graphs. *Networks*, 10:167–178, 1980.

[2] G. Dahl. Directed Steiner problems with connectivity constraints. *Discrete Applied Math.*, 47:109–128, 1993.

[3] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[4] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of ACM*, 34(3):596–615, 1987.

[5] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[6] S. L. Hakimi. Steiner's problem in graphs and its applications. *Networks*, 1:113–133, 1971.

[7] T.-s. Hsu, K.-H. Tsai, D.-W. Wang, and D. T. Lee. Steiner problems on directed acyclic graphs. In J. Y. Cai and C. K. Wong, editors, *Lecture Notes in Computer Science 1090: Proceedings of the 2nd International Symposium on Computing and Combinatorics*, pages 21–30. Springer-Verlag, New York, NY, 1996.

[8] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22:55–89, 1992.

[9] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. Annals of Discrete mathematics 53. North-Holland, 1992.

[10] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, New York, 1976.

[11] S. Martello and P Toth. Finding a minimum equivalent graph of a digraph. *Networks*, 12:89–100, 1982.

[12] L. Nastansky, S. M. Selkow, and N. F. Stewart. Cost-minimal trees in directed acyclic graphs. *Zeitschrift für Operations Research*, pages 59–67, 1974.

[13] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, pages 277–288, 1992.

[14] R. E. Tarjan. *Data Structures and Network Algorithms*. SIAM Press, Philadelphia, PA, 1983.

[15] S. Voss. Worst-case performance of some heuristics for Steiner's problem in directed graphs. *Information Processing Letters*, 48:99–105, 1993.

[16] P. Winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.

[17] R. T. Wong. A dual ascent approach for Steiner tree problems on a directed graphs. *Mathematical Programming*, 28:271–287, 1984.