Approximation Algorithms for (S, T)-Connectivity Problems

by

Bundit Laekhanukit

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We study a directed network design problem called the k-(S,T)-connectivity problem; we design and analyze approximation algorithms and give hardness results. For each positive integer k, the minimum cost k-vertex connected spanning subgraph problem is a special case of the k-(S,T)-connectivity problem. We defer precise statements of the problem and of our results to the introduction.

For $k = 1$, we call the problem the (S,T)-connectivity problem. We study three variants of the problem: the standard (S,T)-connectivity problem, the relaxed (S,T)-connectivity problem, and the unrestricted (S,T)-connectivity problem. We give hardness results for these three variants. We design a 2-approximation algorithm for the standard (S,T)-connectivity problem. We design tight approximation algorithms for the relaxed (S,T)-connectivity problem and one of its special cases.

For any k, we give an $O(\log k \log n)$-approximation algorithm, where n denotes the number of vertices. The approximation guarantee almost matches the best approximation guarantee known for the minimum cost k-vertex connected spanning subgraph problem which is $O(\log k \log \frac{n}{n-k})$ due to Nutov in 2009 [62].

Acknowledgements

I would like to thank all my readers Joseph Cheriyan, Jochen Könemann and Chaitanya Swamy for giving me useful comments and suggestions. I would like to give special thanks to Joseph Cheriyan for supervising and supporting me throughout my Master study.

Dedication

This thesis is dedicated to my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Frank and Jordan in 1995 [38] introduced the following problem on the design of directed networks, based on earlier results of Schrijver [70, 71]. We are given a directed graph $G = (V, E_0)$ and two sets of vertices $S, T \subseteq V$. We may assume that $S \cap T = \emptyset$ (See Chapter 2). The goal is to find the minimum number of edges whose addition to G results in an (S, T)-connected directed graph. In this context, (S, T)-connected means that every pair of vertices $s \in S$ and $t \in T$ is connected by a directed $s \to t$ path. In this setting, we are allowed to add any directed edge that has tail in S and head in T. Frank and Jordan showed that this problem is polynomial-time solvable. However, they did not give a combinatorial algorithm. In fact, they showed that the natural linear programming relaxation of this problem has an integral optimal solution. A combinatorial algorithm for the case $k = 1$ was given by Enni in 1999 [23]. For the case that k can be arbitrary, an algorithm based on the primal-dual scheme was given by Benczúr and Végh [75, 76] in 2005 based on earlier results of Benczúr [8].

Closely related to (S, T)-connectivity problems, the minimum cost k-vertex and k-edge connected spanning subgraph problems in directed graphs have been studied for decades. When $k = 1$ both the vertex and edge connectivity problems are the same as the minimum cost strongly connected subgraph problem, which is NP-Hard. A reduction from the vertex cover problem in cubic graphs was given by Gabow, Goemans, Tardos, and Williamson in [40, 41] in 2005. This shows that these two problems are APX-hard in general and thus have no PTAS. On the algorithmic side, in 1992, Khuller and Vishkin [51, 52] gave a 2-approximation algorithm for the edge-connectivity problem in both directed and undirected graphs. The vertex-connectivity problem seems to be harder than the edge-connectivity problem. In 1995, an $O(\log k)$-approximation algorithm was claimed by Ravi and Williamson [66, 67], but it was later found to be flawed [68, 69]. Exploiting the structure of 3-critically k-connected graph, in 2002, Cheriyan, Vempala and Vetta [15, 16] gave an algorithm with an approximation guarantee of $O(\log k)$ for the special case where $n < 6k^2$ in undirected graphs. In 2005, the result has been generalized by Kortsarz and Nutov [56] to obtain an approximation guarantee of $O(\ln k \cdot \min\{\sqrt{k}, \frac{n}{n-k} \ln k\})$ in both directed and undirected graphs. Three years later, the result has been improved by Fakcharoenphol and Laekhanukit [24] to give an approximation guarantee of $O(\log^2 k)$ which is the first approximation algorithm that achieves

a polylogarithmic approximation guarantee for all values of n and k. In 2009, Nutov [62] added a preprocessing step to the algorithm and improved the approximation guarantee to $O(\log \frac{n}{n-k} \log k)$.

The rooted k-connection problem is similar to the (S,T)-connectivity problem. In this problem, we are given a directed graph with a specified root vertex r and a set of terminals. The goal in this problem is to find a set of edges with minimum cost that makes the directed graph connected from r to every terminal by k-edge disjoint directed paths. The set of positive cost edges in this problem are restricted to have heads in the set of terminals. The most common version of this problem is when all the vertices in the graph except the root vertex are terminals; this is called the optimal branching problem. Between 1960 to 1970, the special case where $k = 1$ has been studied by several researchers. In 1967, Edmonds [22] gave an algorithm for the optimal branching problem. An algorithm based on a linear programming relaxation was given in 1974 by Fulkerson [39]. A faster implementation of the algorithm for the optimal branching problem was given by Tarjan in 1974 [72]. Fulkerson's result has been extended by Frank [34] in 1979. Frank's algorithm is more general and can be applied to the problem of covering an intersecting family by a directed graph. In particular, Frank's algorithm applies to the case where the set of terminals is an arbitrary subset of the vertex set of the directed graph, but every edge of positive cost has its head in the set of terminals. In 1999, Frank [35] gave a polynomial-time algorithm for a special case of the problem. Recently, in 2009, Frank [37] gave another algorithm for the rooted k-connection problem.

The problems studied in this thesis are the minimum cost (S,T)-connectivity problem and the minimum cost k-(S,T)-connectivity problem. In both problems, we are given an initial directed graph and a set of augmenting edges. The augmenting edges are associated with positive costs. The goal in the minimum cost (S,T)-connectivity problem is to find a set of augmenting edges with minimum cost that makes the directed graph (S,T)-connected. Please see Chapter 2 for a precise statement.

1.1 Summary of Results

The following is the summary of our results.

The (S,T)-connectivity Problem ($k = 1$): We consider three variants of the (S,T)-connectivity problem.

- **The standard (S,T)-connectivity problem:** In the first variant, we consider the case where all augmenting edges have tails in S and heads in T. We show that the standard (S,T)-connectivity problem is at least at hard as the minimum cost strongly connected subgraph problem. Our main result for this variant is a 2-approximation algorithm.
- **The relaxed (S,T)-connectivity problem:** In this variant, all augmenting edges have heads in T but there is no restriction on tails. We show that this problem is at least as hard as the directed Steiner tree problem. We design an approximation algorithm and prove that

it achieves the best known approximation guarantee for the directed Steiner tree problem, which is a special case.

- **The relaxed (S,T)-connectivity problem on a digraph that is acyclic on T:** In this variant, we consider the relaxed (S,T)-connectivity problem, where the initial directed graph has no directed circuits containing two vertices of T. We show that this problem is at least as hard as the set cover problem. The main result on this variant is an approximation algorithm whose approximation guarantee matches the approximation threshold of the set cover problem.

- **The unrestricted (S,T)-connectivity problem.** We show that when there is no restriction on augmenting edges, the problem is as hard as the directed Steiner forest problem. In fact, there are reductions in both direction, that is, the unrestricted (S,T)-connectivity problem is a special case of the directed Steiner forest problem, while the latter problem can also be reduced to the former problem.

The k-(S,T)-connectivity problem ($k > 1$): We study the extension of the standard (S,T)-connectivity problem. In the k-(S,T)-connectivity problem, the connectivity requirement between S and T becomes an arbitrary number k. Please see Chapter 3 for a precise statement. We show that the problem is at least as hard as the minimum cost k-vertex connected spanning subgraph problem. Also, we design an approximation algorithm based on the framework of Fakcharoenphol and Laekhanukit [24]. Our approximation algorithm achieves an approximation guarantee of $O(\log n \log k)$. When $k = n^{\Omega(1)}(e.g., k = n^{0.1})$, the approximation guarantee matches the approximation guarantee of the algorithm of Fakcharoenphol and Laekhanukit for the minimum cost k-vertex connected spanning subgraph problem.

1.2 Organization

The organization of this thesis is as follows. In Chapter 2, we discuss the (S,T)-connectivity problem. We present results on three variants of the (S,T)-connectivity problem. In Chapter 3, we discuss the k-(S,T)-connectivity problem. We give a polylogarithmic-approximation algorithm for this problem. In Chapter 4, the last chapter, we give some conclusions and open problems.

Chapter 2

Approximation algorithms for (S, T)-connectivity problems

2.1 Introduction

In this chapter, we discuss the (S, T)-connectivity problem. We study this problem in various settings. The first setting is that all augmenting edges have tails in S and heads in T. In the second setting, augmenting edges still have heads in T, but no restriction is placed on the tails. We also consider the problem in the most general setting when there is no restriction at all. We show that the hardness of this problem depends on the type of augmenting edges. The major contributions in this chapter are approximation algorithms for the first two settings.

2.1.1 Organization

The organization of this chapter is as follows. Section 2.2 has key definitions. In Section 2.3, we start our discussion by giving the hardness results of the (S, T)-connectivity problem. In Section 2.4, we proceed to the case of the standard (S, T)-connectivity problem, where all augmenting edges have tails in S and heads in T. In Section 2.5, we discuss the relaxed (S, T)-connectivity problem, where all augmenting edges have heads in T, but there are no restriction on the tails. In Section 2.6, we discuss the special case of the relaxed (S, T)-connectivity problem where the given digraph is acyclic on T.

2.2 Preliminaries

In the minimum cost (S, T)-connectivity problem, we are given a digraph $G = (V, E_0 \cup E)$, where $E_0 \cap E = \emptyset$, and two sets of vertices S and T. We call $G_0 = (V, E_0)$ the *initial digraph* and call

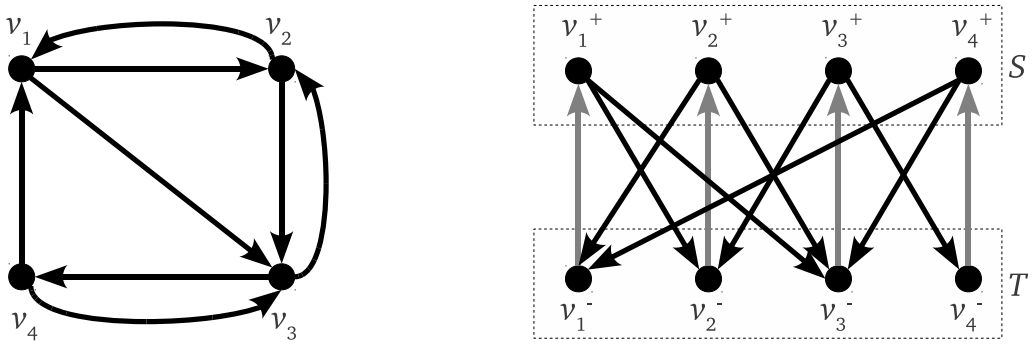

Figure 2.1: The reduction from the instance of the (S,T)-connectivity problem where $S \cap T \neq \emptyset$ to the instance such that $S \cap T = \emptyset$. The left figure is the original instance where $S = T = V$, and the right figure is the transformed instance where $S \cap T = \emptyset$. The original instance is, in fact, the instance of the minimum cost strongly connected subgraph problem. In this figure, the black lines denote positive cost edges, and the grey lines denote zero-cost edges.

edges in E *augmenting edges*. We also have non-negative cost assigned to augmenting edges. We may assume that the edges in E_0 have zero-cost while the edges in E have positive cost. The set of vertices $S \cup T$ might not contain all vertices. We call vertices that are in $V - (S \cup T)$ *optional vertices*. We say that a digraph is (S,T)-*connected* if there is an $s \to t$-dipath connecting every vertex $s \in S$ and $t \in T$. The goal in this problems is to find a minimum cost subset of edges $E' \subseteq E$ so that the digraph $G' = (V, E_0 \cup E')$ is (S,T)-connected.

We call the problem when augmenting edges are restricted to have tails in S and heads in T the *standard* (S,T)*-connectivity problem*. If all augmenting edges have heads in T, there are no restriction on the tails, then we call the problem the *relaxed* (S,T)*-connectivity problem*. In the most general version, when there is no restriction on augmenting edges, then we call it the *unrestricted* (S,T)*-connectivity problem*.

The next result shows that there is no loss of generality in assuming that S and T are disjoint.

Proposition 2.2.1. *There is a reduction from instances of the* (S,T)*-connectivity problem where* $S \cap T \neq \emptyset$ *to instances such that* $S \cap T = \emptyset$ *that preserves the feasibility and the cost of solutions.*

Proof. For each vertex $v \in S \cap T$, we split v into two vertices v^+ and v^-; and join them by an edge (v^-, v^+) with zero-cost. For all edges having v as tails (resp. heads), we change their tails (resp. heads) to v^+ (resp. v^-). Finally, we include v^+ to S and v^- to T for all $v \in S \cap T$. Since all positive cost edges still have tails in S and heads in T, the reduction preserves the restriction on augmenting edges. The reduction is illustrated in Figure 2.1.

Notice that there is a one-to-one mapping between positive cost edges of the original and the transformed instances. We can map any dipath in the original instance to a dipath in the transformed one by replacing a vertex $v \in S \cap T$ by a subpath $v^- \to v^+$. This does not increase the cost because an edge (v^-, v^+) has zero-cost. Conversely, we can map any dipath in the transformed instance to that in the original one by replacing subpath $v^- \to v^+$ (or v^+ if it is the start vertex) by a single vertex v. Hence, the solution to the original instance is feasible if and only if its corresponding solution is feasible to the transformed instance. Moreover, the cost of the optimal solutions are the same in both instances. □

Problems that are related to the (S, T)-connectivity problem are the *directed Steiner tree problem* and the *directed Steiner forest problem*. In the directed Steiner tree problem, we are given a digraph $G = (V, E)$ with a non-negative cost $\mathrm{c}(e)$ on each edge $e \in E$, a root vertex $r \in V$, and a set of terminals $T \subseteq V - \{r\}$. There are two versions of the directed Steiner tree problem: the in-directed Steiner tree problem and the out-directed Steiner tree problem. The goal in the in-directed Steiner tree problem is to find a subgraph $F \subseteq G$ with minimum cost so that F has a dipath from every terminal $t \in T$ to a root vertex r. Note that a minimal solution subgraph is acyclic. In the out-directed Steiner tree problem, F is required to have a dipath from r to every terminal $t \in T$. The directed Steiner forest problem is a generalization of the directed Steiner tree problem. In the directed Steiner forest problem, we are give a digraph $G = (V, E)$ with a non-negative cost $\mathrm{c}(e)$ on each edge $e \in E$ and a set of demands $D \subseteq V \times V$. The goal in the directed Steiner forest problem is to find a minimum cost set of edges $E' \subseteq E$ so that $G' = (V, E')$ has an $s \to t$ dipath for every demand pair $(s, t) \in D$. In general, G' may contain dicycles. However, in the setting of undirected graphs, a minimal solution subgraph is acyclic, that is, G' is a forest. Hence, the problem is called the Steiner forest problem.

At the end of this section, we give the known hardness results of the minimum cost strongly connected subgraph problem, the directed Steiner tree problem, and the directed Steiner forest problem.

Theorem 2.2.2 ([41, 49]). *The minimum cost strongly connected subgraph problem is* APX*-hard.*

Theorem 2.2.3 ([46]). *For every fixed $\epsilon > 0$, the directed Steiner tree problem cannot be approximated within a ratio of $\Omega(\log^{2-\epsilon} n)$ unless $\mathrm{NP} \subseteq \mathrm{ZPTIME}(n^{\mathrm{polylog}(n)})$.*

Theorem 2.2.4 ([21]). *For every fixed $\epsilon > 0$, the directed Steiner forest problem cannot be approximated within a ratio of $\Omega(2^{\log^{1-\epsilon} n})$ unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\mathrm{polylog}(n)})$.*

Throughout this chapter, we use n and m to denote the number of vertices and the number of edges, respectively. We denote the set of edges in an optimal solution by E^* and denote its cost by opt.

2.3 The hardness of the (S,T)-connectivity problem

The hardness of the (S,T)-connectivity problem depends on the type of augmenting edges. We will give reductions from three versions of the (S,T)-connectivity problem to other problems whose hardness have been studied. Our reductions are *approximation-preserving*, that is, the reductions preserve both feasibility and the cost of a solution. The following theorem states the hardness results of the three versions of the (S,T)-connectivity problem.

Theorem 2.3.1. *The hardness of (S,T)-connectivity problem depends on the type of augmenting edges.*

- **The standard (S,T)-connectivity problem:** *If all augmenting edges have tails in S and heads in T, then the problem is at least as hard as the minimum cost strongly connected subgraph problem.*
- **The relaxed (S,T)-connectivity problem:** *If all augmenting edges have heads in T, but there is no restriction on their tails, then the problem is at least as hard as the directed Steiner tree problem.*
- **The unrestricted (S,T)-connectivity problem:** *If there is no restriction on augmenting edges, then the problem is at least as hard as the directed Steiner forest problem.*

Proof. We will describe the hardness construction of each version of the problem. We recall that an instance of (S,T)-connectivity problem consists of a digraph $G = (V, E_0 \cup E)$, sets of vertices S and T, and positive cost $\mathrm{c}(e)$ on each augmenting edge $e \in E$.

The standard (S,T)-connectivity problem: The reduction from the minimum cost strongly connected subgraph problem to the standard (S,T)-connectivity problem is straightforward. Let $G = (V,E)$ be a digraph of the instance of the minimum cost strongly connected subgraph problem. We form the instance of the standard (S,T)-connectivity problem by setting $S = T = V$ and setting $E_0 = \emptyset$. It is clear that the restriction on heads and tails of augmenting edges holds because $S = T = V$. In fact, the minimum cost strongly connected subgraph problem is a special case of the standard (S,T)-connectivity problem. Moreover, by Proposition 2.2.1, we can transform this instance to an instance such that $S \cap T = \emptyset$.

The relaxed (S,T)-connectivity problem: The reduction from the directed Steiner tree problem is as follows. The given instance of the in-directed Steiner tree problem (See Section 2.2.) consists of a digraph $\hat{G} = (\hat{V}, \hat{E})$ with non-negative cost on edges, a root vertex $r \in V$ and a set of terminals $\hat{S} \subseteq \hat{V} - \{r\}$. We may assume that each terminal $s \in \hat{S}$ is incident to a unique edge which is outgoing from s and has zero-cost. Otherwise, we can replace each terminal $s \in \hat{S}$ by a dummy terminal s' and attach s' to s by a zero-cost edge (s', s). Observe that the reduction does not increase

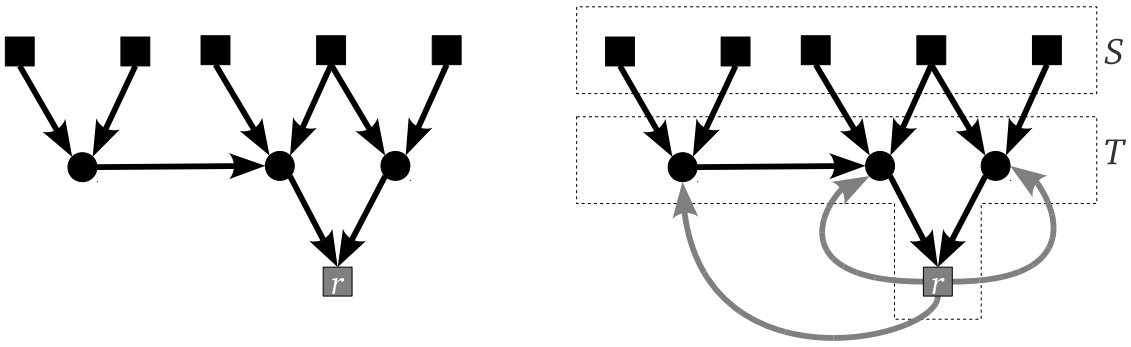

Figure 2.2: The reduction from the instance of the directed Steiner tree problem to the instance of the relaxed (S, T)-connectivity problem. The left figure is the instance of the former problem with the root vertex r. The squares denote terminals, and the circles denote Steiner vertices. The right figure is the instance of the relaxed (S, T)-connectivity problem. The black lines denote positive cost edges, and the grey lines denote zero-cost edges.

the cost nor violate the feasibility of an optimal solution. We use $\hat{G}$ as the base construction and add auxiliary edges with zero-cost from the root vertex r to all non-terminal vertices. Precisely, the digraph of the instance of the relaxed (S, T)-connectivity problem is $G = (V, E_0 \cup E)$, where $V = \hat{V}$ and $E_0 \cup E = \hat{E} \cup \{(r, v) : v \in V - \hat{S}\}$. We define S to be the set of terminals $\hat{S}$ and T to be the set of non-terminal vertices, that is, $S = \hat{S}$ and $T = V - S$. Note that T also includes the root vertex r. We define the set of edges E_0 of the initial digraph to be the set of all zero-cost edges including the auxiliary ones. We define the set of augmenting edges E to be the set of all positive cost edges. The construction is valid for the relaxed (S, T)-connectivity problem because all positive cost edges have heads in T, the set of non-terminal vertices. The reduction is illustrated in Figure 2.2.

The mapping between a solution of the directed Steiner tree problem and that of the relaxed (S, T)-connectivity problem is straightforward. We can transform the solution of the directed Steiner tree problem to that of the relaxed (S, T)-connectivity problem by adding auxiliary edges. Conversely, we can transform the solution of the relaxed (S, T)-connectivity problem to that of the directed Steiner tree problem by removing auxiliary edges. Observe that, for all vertices $s \in S$ and $t \in T$, we can extend any $s \to r$ dipath to an $s \to t$ dipath by adding an auxiliary edge (r, t). This means that any feasible solution to the directed Steiner tree problem is also feasible to the relaxed (S, T)-connectivity problem. Conversely, for all vertices $s \in S$, any simple $s \to r$ dipath contains no auxiliary edges because all auxiliary edges are leaving r. This means that removing auxiliary and redundant edges from the (S, T)-connected subgraph gives us a directed Steiner tree. Thus, the reduction is approximation-preserving.

The unrestricted (S,T)-connectivity problem: The reduction from the directed Steiner forest problem is as follows. The given instance of the directed Steiner forest problem (See Section 2.2.) consists of a digraph $\hat{G} = (\hat{V}, \hat{E})$ with non-negative costs on edges, and a set of demands $D \subseteq V \times V$. We may assume that there are a set of sources S and a set of sinks T, where $S \cap T = \emptyset$. Moreover, we may assume that each source $s \in S$ is incident to one outgoing edge but incident to no incoming edges. Similarly, we may assume that each sink $t \in T$ is incident to one incoming edge but incident to no outgoing edges. The reduction can be done similarly to that of the directed Steiner tree problem. For each source $s \in S$, we add a dummy vertex s' and attach it to s by a zero-cost edge (s', s). Likewise, for each sink $t \in T$, we add a dummy vertex t' and attach it to t by a zero-cost edge (t, t'). We then replace the demand (s, t) by (s', t') for all $(s, t) \in D$. Since the dummy sources and sinks are attached to the original ones by zero-cost edges, the reduction does not increase nor violate the feasibility. Note that each source and sink may occur in more than one demands, e.g., we may have both (s, t_1) and (s, t_2) in D. To construct an instance of the unrestricted (S,T)-connectivity problem, we start with the digraph $\hat{G}$. We define S and T to be the set of sources and sinks, respectively. For all ordered pairs (s, t) with $s \in S$ and $t \in T$, if $(s, t) \notin D$, then we add an auxiliary edge (s, t) with zero-cost. In other words, we pad the digraph with auxiliary edges to handle the new demands implicit in the unrestricted (S,T)-connectivity problem. The set of edges E_0 is defined to be the set of all zero-cost edges including the auxiliary ones. The set of augmenting edges E is defined to be the set of positive cost edges. Clearly, the construction is valid for the unrestricted (S,T)-connectivity problem because there is no restriction on augmenting edges. The reduction is illustrated in Figure 2.3.

Given a solution to the directed Steiner forest problem, we can transform it to a solution to the unrestricted (S,T)-connectivity problem by adding auxiliary edges. Conversely, given a solution to the unrestricted (S,T)-connectivity problem, we can transform it to a solution to the directed Steiner forest problem by removing auxiliary edges. The cost of the two solutions are the same because all auxiliary edges have zero-cost. Consider any feasible solution $\hat{G}'$ to the directed Steiner forest problem. Since all demands are satisfied, there is an $s \to t$ dipath for all $(s, t) \in D$. Moreover, in the corresponding solution G' to the unrestricted (S,T)-connectivity problem, there must be an auxiliary edge (s, t) for all $(s, t) \notin D$. This implies that G' is (S,T)-connected. In the other direction, consider any feasible solution G' to the unrestricted (S,T)-connectivity problem. Observe that, for all $(s, t) \in D$, any $s \to t$ dipath contains no auxiliary edges. This is because a source $s \in S$ has no incoming edges, and a sink $t \in T$ has no outgoing edges. This means that, for all $(s, t) \in D$, an $s \to t$ dipath is still available after removing auxiliary edges. Thus, the corresponding solution $\hat{G}'$ is feasible to the directed Steiner forest problem implying that the reduction is approximation-preserving. □

2.4 The standard (S,T)-connectivity problem.

In this section, we discuss the standard (S,T)-connectivity problem where augmenting edges all have tails in S and heads in T. In general, this problem is at least as hard as the minimum cost

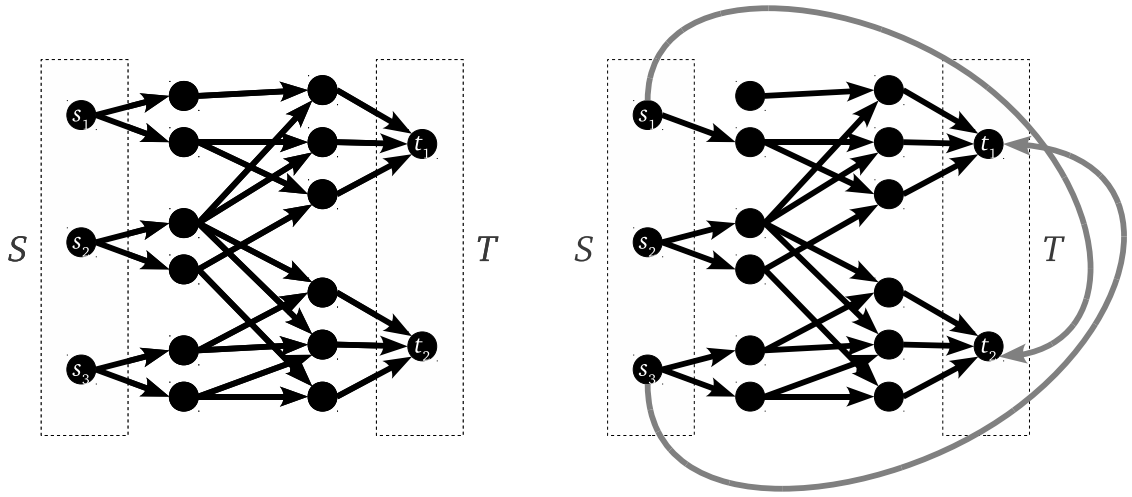

Figure 2.3: The reduction from the instance of the directed Steiner forest problem to the instance of the unrestricted (S, T)-connectivity problem. The left figure is the instance of the former problem. The set of demands in this instance is $\{(s_1, t_1), (s_2, t_1), (s_2, t_2), (s_3, t_2)\}$. The right figure is the instance of the unrestricted (S, T)-connectivity problem. The black lines denote positive cost edges, and the grey lines denote zero-cost edges.

strongly connected subgraph problem. While we have a simple 2-approximation for the minimum cost strongly connected subgraph problem, the interesting question is whether the standard (S,T)-connectivity problem admits a 2-approximation. The answer is yes, and we will answer this question by presenting the 2-approximation algorithm for this problem. Our algorithm is the generalization of the 2-approximation algorithm for the minimum cost strongly connected subgraph problem.

To get some insight, we will briefly describe the 2-approximation algorithm for the minimum cost strongly connected subgraph problem. The algorithm first chooses an arbitrary root vertex r and then finds a minimum cost in-branching J^{in} and a minimum cost out-branching J^{out} rooted at the same vertex r. The solution to the minimum cost strongly connected subgraph problem is the union of J^{in} and J^{out}. Observe that the solution is feasible. To see that, consider any two vertices $u, v \in V$. By the construction, we have $u \to r$ dipath in J^{in} and $r \to v$ dipath in J^{out}. Concatenating these dipaths, we have $u \to v$ dipath for all $u, v \in V$. Thus, the digraph $J^{in} \cup J^{out}$ is strongly connected. Moreover, since any optimal solution to the minimum cost strongly connected subgraph problem must contain both in-branching and out-branching rooted at r, we have that $\mathrm{c}(J^{in} \cup J^{out}) \leq 2\mathrm{opt}$.

We will now describe our algorithm. Consider the standard (S,T)-connectivity problem in two cases.

1. There is no $T \to S$ dipath in the initial digraph.
2. There exists an $T \to S$ dipath in the initial digraph.

We will show that the first case is polynomial-time solvable while the second case admits a 2-approximation. The algorithm in the second case is similar to that of the minimum cost strongly connected subgraph problem.

2.4.1 A polynomial-time algorithm for the case that there is no $T \to S$ dipath

In this section, we assume that the given initial digraph $G_0 = (V, E_0)$ has no $T \to S$ dipath. The following two-phase algorithm shows that this case can be solved in polynomial time.

Phase 1: Reduction In the first phase, we apply some reductions to the initial digraph $G_0 = (V, E_0)$. Intuitively, we want to preprocess the instance by eliminating some terminals from S and T that can share the same connections. This follows by the following two observations.

First, consider any strongly connected component C of G_0. Suppose there is an $s \to t$ dipath for some vertex $s \in C$ and t. Then, for any vertex $s' \in C$, we also have $s' \to t$ dipath. This is because we can extend the $s \to t$ dipath by $s' \to s$ dipath of C. Moreover, this does not increase the cost of the dipath because all edges of C have zero-cost. In short, any two vertices of S lying

in the same strongly connected component of G_0 can share the same connections to T. So, we may think of C as a single vertex. This also applies to a strongly connected component on vertices of T. Note that any strongly connected component cannot contain both S and T vertices because there is no $T \to S$ dipath.

Second, consider any vertices $s, s' \in S$. Suppose there is an $s \to s'$ dipath in G_0. Then this dipath has zero-cost by the definition. So, we can extend any $s' \to T$ dipath to be $s \to T$ dipath without increasing the cost. The same observation also applies to vertices $t, t' \in T$ that have $t' \to t$ dipath. This means that we can *deactivate* some S and T vertices, that is, we remove vertices from S or T and make them optional vertices. Note that deactivated vertices are not removed from the digraph. Indeed, we deactivate vertices of S and T that can share connections with the others.

Our reduction is straightforward from the observations.

- We first contract all strongly connected components.
- We then deactivate some S and T vertices. Note that we also remove augmenting edges whose heads or tails became optional vertices. The conditions for deactivating S and T vertices are as follows.
  - Deactivate a vertex $s \in S$ if there is an $s \to (S - \{s\})$ dipath; in other words, we deactivate all S vertices that have dipaths connecting to other S vertices.
  - Deactivate a vertex $t \in T$, if there is a $(T - \{t\}) \to t$ dipath; in other words, we deactivate all T vertices that are reachable from other T vertices.

After the reduction, the reduced instance has no dicycles. Moreover, there is no dipath between S vertices, and there is no dipath between T vertices.

Phase 2: Connecting Given S and T vertices remaining from the first phase, for each $s \in S$ and $t \in T$, we simply choose an augmenting edge $(s, t) \in E$ if there is no $s \to t$ dipath in the initial digraph. We claim that this augmenting edge always exists.

Figure 2.4 illustrates the working of our algorithm.

Correctness

We will show that our algorithm yields an optimal solution. The followings two lemmas show that the chosen edges are sufficient and necessary. Without loss of generality, we may assume that there is no strongly connected component in the initial digraph $G_0 = (V, E_0)$. To see that, consider any two vertices $u, v \in V$ lying on the same strongly connected component C. Observe that there is a $u \to w$ dipath in G_0 if and only if there is a $v \to w$ dipath. This is because we can extend one dipath to the other by a zero-cost dipath in C. So, we may think of C as a single vertex. We refer to a digraph after the reduction by $\hat{G} = (V, E_0 \cup \hat{E})$. We denote the set of S and T vertices that

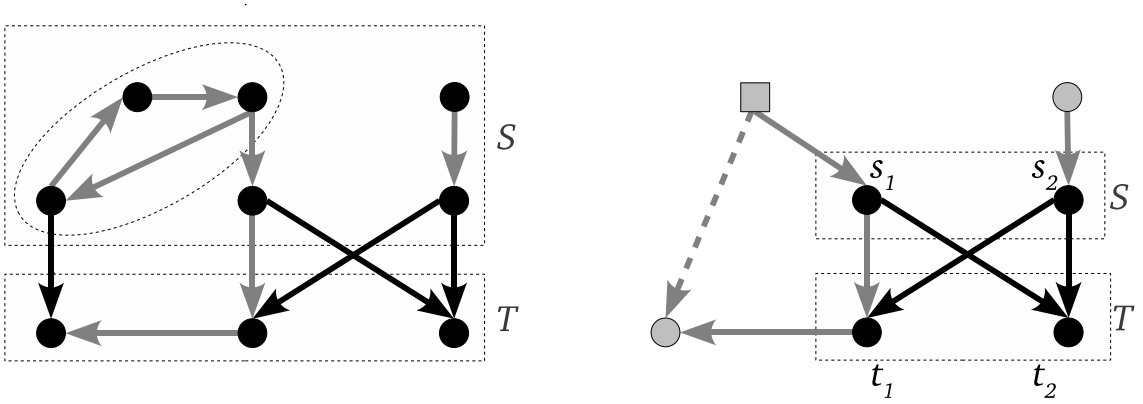

Figure 2.4: The figure illustrates the working of our algorithm for the standard (S, T)-connectivity problem on an example that has no $T \to S$ dipath. The left figure shows the original digraph. The right figure shows the digraph after the reduction. The grey lines denote the edges of the initial digraph $G_0 = (V, E_0)$. The black lines denote the augmenting edges. The square in the right figure denote the vertex obtained by contracting the strongly connected component in the left figure. The light grey vertices denote S and T vertices that are deactivated. The dash-line denote the augmenting edges that are removed. In this example, to make the digraph (S, T)-connected, we have to choose augmenting edges (s_1, t_2), (s_2, t_1), and (s_2, t_2).

are not deactivated in the first phase by $\hat{S}$ and $\hat{T}$, respectively. Note that some augmenting edges of E might not be available in $\hat{E}$.

The following lemma shows the feasibility of the solution.

Lemma 2.4.1. *Consider a subgraph G' of $\hat{G}$. If G' is $(\hat{S}, \hat{T})$-connected, then G' is (S, T)-connected.*

Proof. We will show that there is a dipath from every vertex $s \in S$ to every vertex $t \in T$ in G'. First, for each vertex $s \in S - \hat{S}$, there must be a dipath from s to some vertex $\hat{s} \in \hat{S}$. This is because of the deactivating conditions. Similarly, for each vertex $t \in T - \hat{T}$, there must be a dipath from some vertex $\hat{t} \in \hat{T}$ to t. Since we assume that G' is $(\hat{S}, \hat{T})$-connected, that is, there is an $\hat{s} \to \hat{t}$ dipath connecting every pair of vertices $\hat{s} \in \hat{S}$ and $\hat{t} \in \hat{T}$, we have an $S \to T$ dipath of the form

$$s \to \ldots \to \hat{s} \to \ldots \to \hat{t} \to \ldots \to t.$$

This implies that the digraph G' is (S, T)-connected as claimed. □

The following lemma shows that if the given instance is feasible, then our algorithm always gives a feasible solution. As a consequence, this implies the optimality of the solution.

Lemma 2.4.2. *Suppose the given instance is feasible. For any vertex $\hat{s} \in \hat{S}$ and $\hat{t} \in \hat{T}$, if there is no $\hat{s} \to \hat{t}$ dipath connecting $\hat{s}$ and $\hat{t}$ in the initial digraph $G_0 = (V, E_0)$, then there must exist an augmenting edge $(\hat{s}, \hat{t}) \in E$. Moreover, $(\hat{s}, \hat{t})$ is the unique edge that connects $\hat{s}$ to $\hat{t}$.*

Proof. Suppose there is no $\hat{s} \to \hat{t}$ dipath in the initial digraph. Consider the original digraph $G = (V, E_0 \cup E)$. Since the given instance is feasible, G must have an $\hat{s} \to \hat{t}$ dipath. Let P be any $\hat{s} \to \hat{t}$ dipath in G. In the standard (S, T)-connectivity problem, all augmenting edges have tails in S and heads in T. Since the initial digraph G_0 has no $\hat{s} \to \hat{t}$ dipath, P must contain an augmenting edge (s, t) joining a pair of vertices $s \in S$ and $t \in T$. Observe that P contains no subpath going from T to S. Otherwise, let's take a shortest $T \to S$ subpath P' of P. Then we have that P' enters S once and never leaves. Thus, P' has no augmenting edges which means that P' is a $T \to S$ dipath in the initial digraph $G_0 = (V, E_0)$, a contradiction. Since all augmenting edges go from S to T, this implies that (s, t) is the unique augmenting edge in P. Consequently, an $\hat{s} \to t$ subpath and a $t \to \hat{t}$ subpath of P must be in the initial digraph G_0. But, then by the deactivating conditions, we have $s = \hat{s}$ and $t = \hat{t}$; otherwise, they would have been deactivated. Hence, $\hat{G}$ has the augmenting edge $(\hat{s}, \hat{t})$, and it must be the unique augmenting edge that connects $\hat{s}$ to $\hat{t}$ proving the lemma. □

Corollary 2.4.3. *Suppose the initial digraph contains no $T \to S$ dipath. Then the standard (S, T)-connectivity problem is polynomial-time solvable. In particular, the algorithm given above yields an optimal solution.*

Proof. The statement follows from Lemma 2.4.2 because each pair of vertices (s, t), $s \in \hat{S}$ and $t \in \hat{T}$, has to be connected directly by an augmenting edge (s, t). The running time is polynomial on m and n because we only need to compute a transitive closure and exhaustively connect every pair of vertices $\hat{s} \in \hat{S}$ and $\hat{t} \in \hat{T}$. □

2.4.2 A 2-approximation algorithm for the case there exists $T \to S$ dipath

In this section, we consider the standard (S, T)-connectivity problem when the initial digraph has $T \to S$ dipath. We will present a 2-approximation algorithm for this case using an idea similar to that of the minimum cost strongly connected subgraph problem.

Our algorithm needs a subroutine for solving a rooted version of the (S, T)-connectivity problem called *rooted connection problem*. The problem is, in fact, a special case of the (S, T)-connectivity problem where S consists of a single vertex r called *root*. We call vertices of T *terminals*. When there is no restriction on the set of augmenting edges, the problem is the same as the directed Steiner tree problem. However, if augmenting edges are restricted to have heads in T, then the problem is polynomial-time solvable. We note that there are two versions of the problem: in and out versions. Given a root vertex r, we say that a digraph is *out-connected to T* if there is $r \to t$ dipath connecting r to every vertex $t \in T$. Similarly, we say that a digraph is *in-connected from T* if there is $t \to r$ dipath connecting every vertex $t \in T$ to r. The out-version of

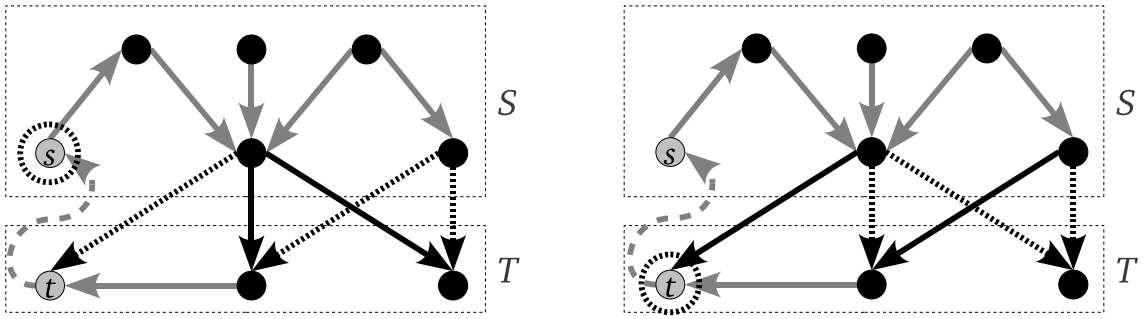

Figure 2.5: The figure illustrates the working of our 2-approximation algorithm for the standard (S,T)-connectivity problem on an example that has a $T \to S$ dipath. The left figure shows the instance of the out-rooted connection problem with the root vertex s and the set of terminals T. The right figure shows the instance of the in-rooted connection problem with the root vertex t and the set of terminals S. The grey lines denote the edges of the initial digraph. The black dash-lines denote the augmenting edges. The grey dash-lines denote the $T \to S$ dipath. The black continuous-lines denote the augmenting edges chosen by the algorithm.

the problem asks to make a digraph out-connected to T while the in-version asks to make a digraph in-connected from T. Unless it is specified, we will refer to the out-version of the problem.

The standard version of the rooted connection problem where all augmenting edges have heads in T can be solved in polynomial time. The results are presented in [34, 35, 37, 33, 7].

Theorem 2.4.4 ([34, 35, 7])**.** *The rooted connection problem where all augmenting edges have heads in T can be solved in $O(m(n+m))$ time.*

We will now describe our algorithm for the (S,T)-connectivity problem, where the initial digraph $G_0 = (V, E_0)$ has a $T \to S$ dipath. We first find a $T \to S$ dipath. Along this dipath, we choose two vertices $s^* \in S$ and $t^* \in T$ and solve two instances of the rooted connection problem. To be precise, in the first subproblem, we take s^* as root, T as the set of terminals and solve the out-rooted connection problem. In the second one, we take t^* as root, S as the set of terminals and solve the in-rooted connection problem. Our solution is the union of augmenting edges from the solutions of the two subproblems.

Figure 2.5 illustrates the working of our algorithm.

Correctness

The feasibility of the solution obtained by our algorithm is shown by the following lemma.

Lemma 2.4.5. *At the termination of the algorithm, the resulting digraph is (S,T)-connected.*

Proof. Let $s^* \in S$ and $t^* \in T$ be vertices in the $T \to S$ dipath chosen by the algorithm. By solving the two subproblems, we have that the resulting digraph is out-connected from s^* to S and in-connected from t^* to T. This implies that we can go from any vertex $s \in S$ to t^*. Likewise, we can go from s^* to every vertex $t \in T$. Since the initial digraph has a $t^* \to s^*$ dipath, we have $s \to t$ dipath of the form

$$s \to \ldots \to t^* \to \ldots \to s^* \to \ldots \to t.$$

Therefore, the resulting digraph is (S, T)-connected as claimed. □

We will now show that our algorithm runs in polynomial time and gives an approximation ratio of 2.

Lemma 2.4.6. *The algorithm given in Section 2.4.2 runs in polynomial time and gives an approximation ratio of* 2*.*

Proof. The first claim follows from the fact that we can find a $T \to S$ dipath and solve two subproblems in polynomial time which is stated in Theorem 2.4.4.

The analysis for the algorithm is straightforward. We first note that a set of edges that is feasible to (S, T)-connectivity problem is also feasible to the two rooted subproblems. So, the cost of the optimal solution of each subproblem has cost at most that of (S, T)-connectivity problem. Thus, the union of these two solutions has cost at most $2\mathrm{opt}$. □

The above two lemmas imply the following theorem.

Corollary 2.4.7. *There exists* 2*-approximation algorithm for* (S, T)*-connectivity.*

2.4.3 General Case

In the general case, we first run a depth first search algorithm to find $T \to S$ dipath. If there is no such dipath, then we run the algorithm given in Section 2.4.1. Otherwise, we run the 2-approximation algorithm given in Section 2.4.2. Combining these two cases we have a 2-approximation algorithm for the general case.

Theorem 2.4.8. *There is a* 2*-approximation algorithm for the* (S, T)*-connectivity problem where all augmenting edges are restricted to have tails in* S *and heads in* T*.*

2.4.4 Running Time

In this paragraph, we analyze the running time of our algorithm. First, we have to run the depth first search algorithm to check whether there is a $T \to S$ dipath. This initial process requires $O(n+m)$ time. In the first case that the initial digraph has no $T \to S$ dipath, the running time is dominated

by the time for computing the reduction. This requires the computation of transitive closure which takes $O(n^3)$ time. In the second case, the running time is dominated by the running time for solving two instances of the rooted connection problem. By Theorem 2.4.4, this requires $O(m(n+m))$ times. Hence, the total running time of our algorithm is $O((n+m)+n^3+m(n+m)) = O(n^3+m^2)$.

2.5 The relaxed (S,T)-connectivity problem

In this section, we will discuss the relaxed (S,T)-connectivity problem. We will characterize the properties of the optimal solution and give an approximation algorithm for this problem. Our algorithm achieves an approximation ratio of $\alpha(n)+1$, where $\alpha(n)$ is the approximation ratio for approximating the directed Steiner tree problem.

2.5.1 Preliminary

Throughout this section, we will use the following definitions.

- *An in-tree (or in-branching) J^{in} rooted at r* is a digraph such that there is a $v \to r$-dipath to r from every vertex $v \in V(J^{in}) - \{r\}$.
- *An out-tree (or out-branching) J^{out} rooted at r* is a digraph such that there is an $r \to v$-dipath from r to every vertex $v \in V(J^{out}) - \{r\}$.
- *A junction tree J rooted at r* is a union of in-tree J^{in} and out-tree J^{out} rooted at the same vertex r. Note that the in-tree and the out-tree may have common edges.
- A digraph G is *acyclic on T* if there is no dicircuit (that is, connected di-Eulerain subgraph) in G containing two distinct vertices of T. In other words, for any two vertices $t, t' \in T$, G either has a $t \to t'$ dipath or a $t' \to t$ dipath but not both. Note that G may have neither a $t \to t'$ dipath nor a $t' \to t$ dipath.

Remark that the definitions of junction tree, in-tree and out-tree are taken from [14]. We need a subroutine that solves the minimum cost branching problem and a subroutine that solves the rooted connection problem. We also need a subroutine that approximately solves the directed Steiner tree problem. The result on the minimum cost branching problem is usually attributed to Edmonds [22]. In fact, the problem can be reduced to the matroid intersection problem.

Theorem 2.5.1 ([7, 22, 72, 39, 34, 35, 37])**.** *There exists a polynomial-time algorithm that finds a minimum cost in-branching (respectively, out-branching).*

Theorem 2.5.2 ([34, 37])**.** *There exists a polynomial-time algorithm that solves the out-rooted (respectively, in-rooted) connection problem optimally, where all positive cost edges have heads (respectively, tails) in the terminal set.*

Theorem 2.5.3 ([11, 9])**.** *There exists an $O(\log^3 n)$-approximation algorithm for the directed Steiner tree problem that runs in quasi-polynomial time.*

Note that the original paper [11] claimed that there exists an $O(\log^2 n)$-approximation algorithm for the directed Steiner tree problem that runs in quasi-polynomial time. However, the authors applied Theorem 2 in [78] which has a gap. The gap was fixed in [47]. After fixing the gap, the approximation ratio of the algorithm in [11] becomes $O(\log^3 n)$.

The following theorem states the main result of this section.

Theorem 2.5.4. *There exists a $(\alpha(n)+1)$-approximation algorithm for the relaxed (S,T)-connectivity problem, where $\alpha(n)$ denotes the (best known) approximation guarantee for the directed Steiner tree problem. In particular, there is an $O(\log^3 n)$-approximation algorithm for the relaxed (S,T)-connectivity problem that runs in quasi-polynomial time.*

2.5.2 Overview

We consider the relaxed (S,T)-connectivity problem in two cases. The first case is that there exists $T \to S$ dipath in the initial digraph $G_0 = (V, E_0)$. The second case is that there is no $T \to S$ dipath in G_0.

Remark that there is a $T \to S$ dipath in the initial digraph $G = (V, E_0)$ if and only if there is a $T \to S$ dipath in $G = (V, E_0 \cup E)$. The forward direction is straightforward. To see the other direction, consider an inclusionwise minimal $T \to S$ dipath P of G. The start vertex is $t \in T$ and the end vertex is $s \in S$. Clearly, P has no augmenting edges because all augmenting edges have heads in T. Thus, P is a dipath in G_0.

We can reduce instances of the first case where the initial digraph G_0 has a $T \to S$ dipath to instances of the second case where the initial digraph has no $T \to S$ dipath. The reduction is the same as given in Theorem 2.3.1. For each vertex $s \in S$, we add a new vertex s' and a new edge (s', s) of zero-cost. Clearly, the original digraph has an $s \to t$ dipath, for $s \in S$ and $t \in T$, if and only if the new digraph has an $s' \to t$ dipath. Let S' be the set of newly added vertices. We consider the (S', T)-connectivity problem instead of the original (S, T)-connectivity problem. It is clear that the reduction does not increase the cost nor violate the feasibility of an optimal solution. Moreover, the reduction maintains the restriction that all augmenting edges have heads in T.

Even though we have the above reduction, we present a simple approximation algorithm for the case that the initial digraph G_0 has a $T \to S$ dipath.

2.5.3 An approximation algorithm for the case that there exists a $T \to S$ dipath

Suppose there exists a $T \to S$ dipath P in the initial digraph $G_0 = (V, E_0)$. This case is similar to the one in Section 2.4.1. Let s^* and t^* be vertices of P such that $s^* \in S$ and $t^* \in T$. We solve two

instances of the rooted connection problem. The first instance Π_{s^*} is the instance of the out-rooted connection problem with root s^* and terminal set T. Since all augmenting edges have heads in T, this instance is polynomial-time solvable. (See Theorem 2.5.2.) The second instance Π_{t^*} is the instance of the in-rooted connection problem with root t^* and terminal set S. This instance might have some augmenting edges with tails not in the terminal set S, and thus, it does not fit in the settings in Theorem 2.5.2. Generally, the instance Π_{t^*} is the instance of the in-directed Steiner tree problem which is NP-hard.

Our algorithm simply finds solutions E_{s^*} and E_{t^*} of the instance Π_{s^*} of the rooted connection problem and the instance Π_{t^*} of the directed Steiner tree problem. The algorithm output $E' = E_{s^*} \cup E_{t^*}$ as the solution to the relaxed (S,T)-connectivity problem. Let $G' = (V, E_0 \cup E')$ denote the digraph obtained by our algorithm.

Lemma 2.5.5 (See Lemma 2.4.5)**.** *The resulting digraph $G' = (V, E_0 \cup E')$ is (S,T)-connected.*

The following lemma shows that, our algorithm gives an approximation ratio of $\alpha(n) + 1$, where $\alpha(n)$ is defined in the lemma.

Lemma 2.5.6. *Assume that the initial digraph $G_0 = (V, E_0)$ contains a $T \to S$ dipath. Suppose there exists an $\alpha(n)$-approximation algorithm for the directed Steiner tree problem. Then there exists an $(\alpha(n) + 1)$-approximation algorithm for the relaxed (S,T)-connectivity problem.*

Proof. Consider an optimal solution $G^* = (V, E_0 \cup E^*)$. Note that G^* contains an $s \to t$ dipath for every $s \in S$ and $t \in T$. We conclude that G^* contains a feasible solution to the instance Π_{s^*} of the rooted connection problem. Similarly, G^* contains a feasible solution to the instance Π_{t^*} of the directed Steiner tree problem. We will now compare the cost of E^* to the cost of the solution obtained by our algorithm. Recall that opt denote the cost of the optimal solution G^*. Since the instance Π_{s^*} of the rooted connection problem is polynomial-time solvable, we have $\mathrm{c}(E_{s^*}) \leq \mathrm{opt}$. Since we use an $\alpha(n)$-approximation algorithm for the directed Steiner tree problem, we have $\mathrm{c}(E_{t^*}) \leq \alpha(n)\mathrm{opt}$. Thus, $\mathrm{c}(E') \leq \mathrm{c}(E_{s^*}) + \mathrm{c}(E_{t^*}) \leq (\alpha(n) + 1)\mathrm{opt}$, proving the lemma. □

2.5.4 An approximation algorithm for the case that there is no $T \to S$ dipath.

We will characterize the important properties of the optimal solution $G^* = (V, E_0 \cup E^*)$. Exploiting the properties, we design an $(\alpha(n) + 1)$-approximation algorithm for this problem, where $\alpha(n)$ is the approximation ratio for approximating the directed Steiner tree problem.

2.5.5 Properties of the optimal solution

Consider an optimal solution $G^* = (V, E_0 \cup E^*)$. We will construct junction trees $J_1, J_2, \ldots, J_\ell \subseteq G^*$ with the following properties:

- For $i = 1, 2, \ldots, \ell$, J_i contains S, and J_i has $s \to t$ dipath for all $s \in S$ and all $t \in J_i \cap T$.
- For $i \neq j$, J_i and J_j have no common vertices of T and thus no common augmenting edges.
- $\bigcup_{i=1}^{\ell} J_i$ contains T. In particular, $\bigcup_{i=1}^{\ell} J_i$ is (S, T)-connected.

Intuitively, we want to partition the set of augmenting edges in the optimal solution E^* so that each subset together with E_0 forms a junction tree J_i connecting S and $J_i \cap T$.

We start our construction by contracting all maximal strongly connected components of G^*. Observe that no vertices of S and T are in the same strongly connected component because there is no $T \to S$ dipath. We will abuse the notation and continue using the same symbols for the contracted digraph. At this point, observe that the digraph G^* is acyclic on T. Hence, there must be a vertex $t \in T$ that has no dipath from any vertex in $T - \{t\}$. We call such a vertex a *top-vertex*.

The construction runs in ℓ iterations. In each iteration i, we construct a junction tree J_i whose in-tree contains S and the out-tree contains some vertices of T. We take a top-vertex t_i as the root of the junction tree. Through t_i, the junction tree J_i connects every vertex of S to every vertex of $J_i \cap T$. In other words, t_i is the "bridge" connecting S and $J_i \cap T$ in J_i. This is because t_i is a top-vertex; hence, every $S \to J_i \cap T$ dipath must visit t_i before reaching other vertices of $J_i \cap T$. We then remove from the current digraph vertices of T that are assigned to J_i. We keep repeating the process until all vertices of T are assigned to some junction trees.

To be precise, we start from the digraph $G_0^* = G^*$ and the terminal set $T_0 = T$. At the iteration i, for $i = 1, 2, \ldots, \ell$, we consider the digraph G_{i-1}^* and the terminal set T_{i-1}. We choose a top-vertex t_i as the root of the junction tree J_i which consists of an in-tree J_i^{in} and an out-tree J_i^{out}, that is, $J_i = J_i^{in} \cup J_i^{out}$. The in-tree J_i^{in} is obtained by taking an in-directed Steiner tree in G_{i-1}^* on the terminal set S. The out-tree J_i^{out} is obtained by taking the union of $t_i \to t$ dipaths in G_{i-1}^* for all vertices $t \in T_{i-1}$ reachable from t_i. Once we have the junction tree J_i, we update the digraph G_{i-1}^* and the terminal set T_{i-1} by removing from them all vertices of T assigned to J_i. Thus, we have $G_i^* = G_{i-1}^* - J_i \cap T$ and $T_i = T_{i-1} - J_i \cap T$. We then continue to the next iteration and repeat the process until all vertices of T are assigned to junction trees. In fact, the stopping condition is $T_\ell = \emptyset$, or equivalently, $\bigcup_{i=1}^{\ell} J_i$ contains T.

The following is the key lemma.

Lemma 2.5.7. *At the iteration i, $i = 1, 2, \ldots, \ell$, the digraph G_{i-1}^* is (S, T_{i-1})-connected.*

Proof. We will proceed by induction on i for $i = 1, 2, \ldots, \ell$.

Base case $i = 1$: The base case is trivial because the starting digraph $G_0^* = G^*$ is obtained from the optimal solution by contracting strongly connected components.

Inductive step $i > 1$: Assume that the induction hypothesis holds for some $i \geq 1$. We will prove that the digraph G_i^* is (S, T_i)-connected. Suppose not. Then there exists a pair of vertices $s \in S$ and $t \in T_i$ that have no $s \to t$ dipath in G_i^*. Since G_{i-1}^* is (S, T_{i-1})-connected, this means that every $s \to t$ dipath P in G_{i-1}^* must contain some vertex $t' \in J_i \cap T$. Since the root t_i of J_i is a top-vertex, we conclude that t' is in J_i if and only if t' is reachable from t_i in G_{i-1}^*. Hence, we can concatenate dipath $t_i \to t'$ in J_i and dipath $t' \to t$ in P to form $t_i \to t$ dipath. By the construction of J_i, vertex t must be included in J_i and thus must have been removed, a contradiction. Therefore, G_i^* is (S, T_i)-connected. □

We will now proves that the three properties holds.

Lemma 2.5.8. *The following properties holds for $J_1, J_2, \ldots, J_\ell$.*

- *(i) For $i = 1, 2, \ldots, \ell$, J_i contains S, and J_i has $s \to t$ dipath for all $s \in S$ and all $t \in J_i \cap T$.*
- *(ii) For $i \neq j$, J_i and J_j have no common vertices of T and thus no common augmenting edges.*
- *(iii) $\bigcup_{i=1}^{\ell} J_i$ contains T. In particular, $\bigcup_{i=1}^{\ell} J_i$ is (S, T)-connected.*

Proof. (i) The first property follows from Lemma 2.5.7. In fact, for $i = 1, 2, \ldots, \ell$, since G_{i-1}^* is (S, T_{i-1})-connected, it must contain an in-directed Steiner tree J_i^{in} rooted at t_i on the terminal set S. Clearly, every vertex $t \in J_i^{out} \cap T$ has $t_i \to t$ dipath in $J_i^{out} \subseteq J_i$. Moreover, we claim that t_i is the unique vertex of T in J_i^{in}, that is, $J_i^{in} \cap T = \{t_i\}$. Note that J_i^{in} is an in-directed Steiner tree in G_{i-1}^* rooted at t_i. So, if J_i^{in} contains some other vertex $t' \in T$, then it has a $t' \to t_i$ dipath and so does G_{i-1}^*. This is a contradiction since t_i is a top-vertex of G_{i-1}^*, that is, G_{i-1}^* has no $(T_{i-1} - \{t_i\}) \to t_i$ dipath. (Note that T_{i-1} is the set of vertices of T remaining in G_{i-1}^*.)

Thus, for all $s \in S$ and $t \in J_i \cap T$, we have an $s \to t$ dipath of the form

$$s \to \ldots \to t_i \to \ldots \to t.$$

(ii) The second property holds because, for $i = 1, 2, \ldots, \ell$, we remove all vertices of $J_i \cap T$ from the digraph G_{i-1}^* before proceeding to the next iteration. Moreover, since all augmenting edges have heads in T, no two junction trees have common augmenting edges.

(iii) The last property holds because we stop the construction when $T_\ell = \emptyset$. Hence, by Property (i), we have that $\bigcup_{i=1}^{\ell} J_i$ is (S, T)-connected.

□

At the termination, we uncontract the strongly connected components of G^*. Observe that any two maximal strongly connected components have no common vertices; otherwise, the two would have been merged. So, the junction trees $J_1, J_2, \ldots, J_\ell$ still have no common vertices of T. Thus,

they have no common augmenting edges. We abuse the notation $\mathrm{c}(J_i)$ to denote the summation of the cost of all edges in J_i and the cost of all edges of strongly connected components that are appeared as contracted vertices in J_i. Then we have $\sum_{i=1}^{\ell} \mathrm{c}(J_i) = \mathrm{opt}$.

After uncontracting, if the root vertex of a junction tree becomes a strongly connected component, then we may take any vertex in the component as a root. This will not violate the reachability because all vertices in the same strongly connected component are reachable from each other. However, the in-tree and the out-tree may have common vertices of T in the uncontracted digraph. In more detail, for $i = 1, 2, \ldots, \ell$, consider the junction tree $J_i = J_i^{in} \cup J_i^{out}$. If the root vertex of J_i in the contracted digraph is formed by contracting a strongly connected component C_i, then some of the vertices of C_i may be contained in both J_i^{in} and J_i^{out}.

An approximation algorithm

Our algorithm does the following.

1. For each vertex $t \in T$, approximately compute the minimum cost in-directed Steiner tree F_t in $G = (V, E_0 \cup E)$ rooted at t taking the set of terminals to be S.
2. Remove all edges incident to S, and contract S into a single vertex s^*.
3. For each vertex $t \in T$, add an auxiliary edge (s^*, t) with cost $\mathrm{c}(s^*, t) = \mathrm{c}(F_t)$.
4. Remove all optional vertices $v \in V - (S \cup T)$ that are not reachable from T in $G_0 = (V, E_0)$. Consider any remaining optional vertex v. Observe that any shortest $T \to v$ dipath has zero-cost because it has no edges with heads in T.
5. Compute a minimum cost out-branching M rooted at s^* in the current digraph. Note that the out-branching M contains all vertices of the digraph and has a dipath from s^* to every vertex. In fact, M is an out-directed spanning tree (or arborescence).
6. Uncontract s^*, and for each auxiliary edge $(s^*, t) \in M$, replace the edge by the in-directed Steiner tree F_t. Let $G' = (V, E_0 \cup E')$ denote the resulting digraph.

We claim that $\mathrm{c}(M) \leq (\alpha(n)+1)\mathrm{opt}$, where $\alpha(n)$ is the approximation ratio for approximating the directed Steiner tree problem. Figure 2.6 illustrates the working of our algorithm.

Theorem 2.5.9. *There exists an $(\alpha(n)+1)$-approximation algorithm for the relaxed (S, T)-connectivity problem, where $\alpha(n)$ is the approximation ratio for approximating the directed Steiner tree problem.*

Proof. We will show that our algorithm achieves the claimed approximation ratio.

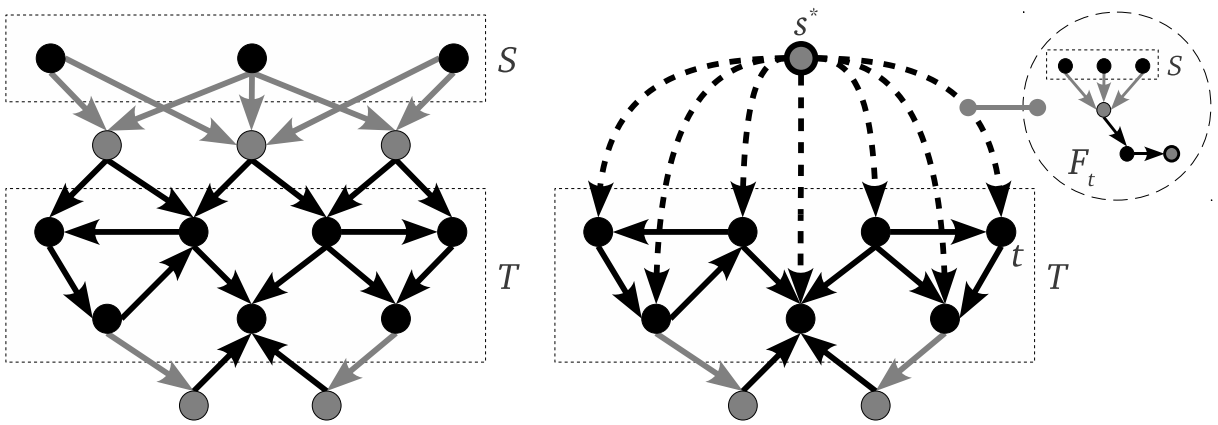

Figure 2.6: The figure illustrates the working of our approximation algorithm for the relaxed (S,T)-connectivity problem on an example that has no $T \to S$ dipath. The left figure shows the digraph before the reduction. The right figure shows the digraph after the reduction. The set of vertices S is contracted into a single vertex s^*. The black vertices are vertices of S and T. The grey vertices are optional vertices. The grey lines denote edges of the initial digraph. The black lines denote augmenting edges. The dashed lines denote auxiliary edges obtained by replacing an in-directed Steiner tree F_t rooted at a root vertex $t \in T$ by an edge (s^*, t) with cost $\mathrm{c}(F_t)$ for each vertex $t \in T$.

Correctness: The correctness of our solution follows from the fact that M is an out-branching. To see that, consider any vertex $t \in T$. Observe that every $s^* \to t$ dipath in M must be of the form

$$s^* \to \hat{t} \to \ldots \to t.$$

where $(s^*, \hat{t})$ is an auxiliary edge while other edges belong to the original digraph. Since we replace $(s^*, \hat{t})$ by the in-directed Steiner tree $F_{\hat{t}}$ in the final step, the resulting digraph G' must have an $s \to \hat{t}$ dipath for every $s \in S$. Hence, we have $s \to t$ dipath of the form

$$s \to \ldots \to \hat{t} \to \ldots \to t.$$

Thus, the resulting digraph G' is (S,T)-connected.

Cost analysis: First, we will construct an upperbound from the digraph $G^* = (V, E_0 \cup E^*)$ of the optimal solution. We construct junction trees $J_1, J_2, \ldots, J_\ell$ from G^* by the construction given Section 2.5.5. Recall that, for $i = 1, 2, \ldots, \ell$, the junction tree J_i is a union of the in-tree J_i^{in} and the out-tree J_i^{out}, where the in-tree J_i^{in} is an in-directed Steiner tree rooted at t_i on the terminal set S.

Similar to the construction in the algorithm, we contract S into a single vertex s^*. For $i = 1, 2, \ldots, \ell$, we replace the in-tree J_i^{in} by an edge (s^*, t_i) with cost $\alpha(n) \cdot \mathrm{c}(J_i^{in})$. By the way we choose the root vertex t_i, there exists a $t_i \to t$ dipath in J_i^{out} for every vertex $t \in J_i^{out} \cap T$.

We include all optional vertices v that are reachable from T to some junction tree. In fact, we include v together with a shortest $T \to v$ dipath. Since any shortest $T \to v$ dipath has no edges with heads in T, this dipath has zero-cost. So, the cost of the junction trees do not increase. We then remove from G^* all optional vertices that are not reachable from T. Clearly, at this point, G^* has a dipath from s^* to every vertex. So, G^* must contain an out-branching rooted at s^* of cost at most that of G^*.

The cost of G^* after the construction might be changed. This is because, in the construction in Section 2.5.5, we contract strongly connected components before constructing junction trees. Consider the contracted digraph in Section 2.5.5. For each junction tree J_i, the in-tree J_i^{in} and the out-tree J^{out} have exactly one vertex of T in common which is the root vertex. If the root vertex of the junction tree J_i is a contracted vertex, then after uncontracting strongly connected components, the in-tree J_i^{in} and the out-tree J_i^{out} may have common augmenting edges. Thus, the cost of G^* after the construction in the above paragraph is at most

$$\sum_{i=1}^{\ell}(\mathrm{c}(J_i^{out}) + \mathrm{c}(J_i^{in})) \leq \sum_{i=1}^{\ell}(\mathrm{c}(J_i) + \mathrm{c}(J_i^{in})).$$

We will now compare the cost of G^* to that of the solution M obtained by our algorithm. Notice that G^* and M have the same set of vertices. Moreover, by the construction, the cost of (s^*, t_i) in M is at most $\alpha(n)$ times that of the minimum cost directed Steiner tree rooted at t_i on the terminal set S. Hence, $\mathrm{c}(s^*, t_i) \leq \alpha(n) \cdot \mathrm{c}(J_i^{in})$ for all $i = 1, 2, \ldots, \ell$. Thus, by the minimality of M, we have

$$\mathrm{c}(M) \leq \sum_{i=1}^{\ell}(\mathrm{c}(J_i)+\alpha(n)\cdot\mathrm{c}(J_i^{in})) \leq \sum_{i=1}^{\ell}(\mathrm{c}(J_i)+\alpha(n)\cdot\mathrm{c}(J_i)) = (\alpha(n)+1)\sum_{i=1}^{\ell}\mathrm{c}(J_i) \leq (\alpha(n)+1)\mathrm{opt}.$$

Note that the last inequality follows from the fact that any two junction trees have no common augmenting edges. This implies that our algorithm yields an approximation ratio of $\alpha(n) + 1$ as required.

□

2.5.6 General case

In both cases, our algorithm achieves an approximation ratio of $\alpha(n) + 1$, where $\alpha(n)$ is the approximation ratio of the algorithm for the directed Steiner tree problem. In the general case, we can run the depth first search algorithm to check which case holds for the instance and apply the appropriate algorithm. In Section 2.3, we showed that the relaxed (S, T)-connectivity problem is at least as hard as the directed Steiner tree problem. This means that our approximation guarantee matches the hardness threshold up to an additive term.

2.6 The relaxed (S,T)-connectivity problem when a digraph is acyclic on T

In this section, we will discuss the special case of the relaxed (S,T)-connectivity problem where the given digraph $G=(V,E_0\cup E)$ is acyclic on T. (See Section 2.5.1.)

Readers who are familiar with the reduction from Set Cover to the directed Steiner tree problem may see that the same reduction applies to the relaxed (S,T)-connectivity problem. This suggests that the problem cannot be approximated within a factor of $(1-\epsilon)\ln n$ [26]. The following results show the hardness of this special case of the relaxed (S,T)-connectivity problem.

Theorem 2.6.1 (Feige'98 [26])**.** *Let $\epsilon>0$. Unless* NP $\subseteq$ DTIME$(n^{O(\log\log n)})$*, there is no $((1-\epsilon)\ln n)$-approximation algorithm for* Set Cover*.*

Theorem 2.6.2. *The special case of relaxed (S,T)-connectivity problem where the given digraph $G=(V,E_0\cup E)$ is acyclic on T is at least as hard as* Set Cover*.*

Proof. The reduction from Set Cover to the relaxed (S,T)-connectivity problem is as follows. The instance of Set Cover consists of a universe $\hat{S}=\{s_1,s_2,\ldots,s_n\}$ and given subsets $S_1,S_2,\ldots,S_m$ with weights $\mathrm{c}(S_1),\mathrm{c}(S_2),\ldots,\mathrm{c}(S_m)$, respectively. We define the vertex set of the relaxed (S,T)-connectivity to be $V=\hat{S}\cup\{S_1,S_2\ldots,S_m\}\cup\{t\}$. Particularly, we create vertices $s_1,s_2,\ldots,s_n$ corresponding to elements of $\hat{S}$, create vertices $S_1,S_2,\ldots,S_m$ corresponding to each subset and create an auxiliary vertex t. We define the set S of the relaxed (S,T)-connectivity problem to be the universe $\hat{S}$ of Set Cover and define $T=\{t\}$. The other vertices of V, namely, $S_1,S_2,\ldots,S_m$ are optional vertices. We add an edge (s_i,S_j) to E_0 if $s_i\in S_j$. We add augmenting edges from each S_j to t with cost equal to the weight of S_j, namely, $\mathrm{c}(S_j)$. It follows by the construction that this instance is valid for the relaxed (S,T)-connectivity problem, and the underlying digraph $G=(V,E_0\cup E)$ is acyclic on T (since T consists of a single vertex).

The construction yields a one-to-one correspondence between subsets of Set Cover and augmenting edges of the (S,T)-connectivity problem. Also, observe that an $S\to t$ dipath must be of the form $s\to S_e\to t$, where $s\in S_e$. This implies that t is reachable from $s\in S_e$ if and only if the augmenting edge (S_e,t) is chosen. So, the solution to the relaxed (S,T)-connectivity problem is feasible if and only if the corresponding solution to Set Cover is feasible. Therefore, the reduction is approximation-preserving, and the lemma follows.

□

Theorem 2.6.2 suggests that this problem cannot be approximated within a factor of $\Omega(\log n)$. The reduction from Set Cover also gives some insight that a greedy algorithm for Set Cover might work well on this special case of the relaxed (S,T)-connectivity problem. Extending this idea, we obtain an approximation algorithm that achieves the approximation guarantee of $O(\log n)$ meeting the lower bound. Let $\alpha'(n)$ be the best known approximation ratio for approximating Set Cover. Then there exists an $\alpha'(n)$-approximation algorithm for the special case of relaxed (S,T)-connectivity problem where the given digraph $G=(V,E\cup E_0)$ is acyclic on T.

Theorem 2.6.3. *There exists an $O(\log n)$-approximation algorithm for the special case of relaxed (S,T)-connectivity problem where the given digraph $G = (V, E \cup E_0)$ is acyclic on T.*

We will present two approximation algorithms for this problem. The first algorithm is the modified version of the the algorithm in Section 2.5.5. The second algorithm exploits the acyclic property of the given digraph $G = (V, E_0 \cup E)$. Both algorithms achieve the same approximation ratio of $\alpha'(n)$, where $\alpha'(n)$ is the approximation ratio for Set Cover .

2.6.1 Approximation algorithm I

Our first algorithm is almost the same as the algorithm in Section 2.5. However, when the given digraph $G = (V, E_0 \cup E)$ is acyclic on T, the characteristic of the optimal solution $G^* = (V, E_0 \cup E^*)$ is slightly different because there is no strongly connected component containing two vertices of T. Hence, in Section 2.5.5, we can construct junction trees $J_1, J_2, \ldots, J_\ell$ from G^* without contracting strongly connected components. So, a root vertex $t_i \in T$ of the junction tree J_i, for $1 \leq i \leq \ell$, is a single vertex. Recall that a top-vertex $t \in T$ is a vertex that has no dipath from $T - \{t\}$. In the construction, we choose a top-vertex as the root vertex of the junction tree. Since all augmenting edges have heads in T, this means that all augmenting edges of the in-tree J_i^{in} must have heads at its root $t_i \in T$. This is an important property because the in-directed Steiner tree problem where all positive cost edges have head at the root vertex is easier than the general case. In fact, the problem can be reduced to Set Cover . We will show the reduction from the directed Steiner tree problem where all positive cost edges have heads at the root vertex to Set Cover .

Theorem 2.6.4. *There is an approximation-preserving reduction between Set Cover and the the in-directed (respectively, out-directed) Steiner tree problem where all positive cost edges have heads (respectively, tail) at the root vertex.*

Proof. By a reduction similar to the reduction in Theorem 2.6.1, we have that the in-directed Steiner tree problem where all positive cost edges have heads at a root vertex is at least as hard as Set Cover . We will show the converse that Set Cover is at least as hard as the in-directed Steiner tree problem where all positive cost edges have heads at the root vertex.

The instance of the in-directed Steiner tree problem consists of a digraph $G = (V, E_0 \cup E)$, where $E_0 \cap E = \emptyset$, with a non-negative cost $\mathrm{c}(e)$ on each edge $e \in E_0 \cup E$, a root vertex $r \in V$, and a set of terminals $T \subseteq V - \{r\}$. The set of edges E_0 is the set of zero-cost edges, and the set of edges E is the set of positive cost edges whose heads are at r. To construct an instance of Set Cover , we take T as the universe. For each edge $e \in E$, we construct a subset S_e with cost $\mathrm{c}(e)$. We add an element $t \in T$ to S_e if there is $t \to r$ dipath in $G' = (V, E_0 \cup \{e\})$, that is, $S_e = \{t \in T : G' = (V, E_0 \cup \{e\})$ contains a $t \to r$ dipath$\}$.

Observe that there is a one-to-one correspondence between a solution to the instance of the in-directed Steiner tree problem and a solution to the instance of Set Cover . Let E^* be the optimal solution to the in-directed Steiner tree problem. Then, by the construction of the instance

of Set Cover , the union of the corresponding subsets of E' is T, that is, $\bigcup_{e \in E'} S_e = T$. Conversely, let $S_{e_1}, S_{e_2}, \ldots, S_{e_\ell}$ be the chosen subsets of the optimal solution to Set Cover . Let $E^* = \{e_1, e_2, \ldots, e_\ell\}$. By the construction, $G' = (V, E_0 \cup E^*)$ has a $t \to r$ dipath for all $t \in T$. We can make G' acyclic by removing unnecessary edges. Precisely, we remove an edge e from G' if $G' - \{e\}$ still has a $t \to r$ dipath for all $t \in T$. By the minimality of E^*, no positive cost edges are removed from G'. Thus, the cost of G' is the same as the cost of E^*. □

An algorithm

The algorithm is the same as the algorithm in Section 2.5.5. Recall that the algorithm starts by constructing an auxiliary digraph as follows. Firstly, for each vertex $t \in T$, we compute an in-directed Steiner tree J_t with the root vertex t and the terminal set S. We then contract S to a single vertex s^* and replace each in-directed Steiner tree J_t by an auxiliary edge (s^*, t) with the same cost as J_t. Finally, we remove all optional vertices that are not reachable from T. After we have an auxiliary digraph, we compute the minimum cost out-branching M with the root vertex s^* and the terminal set T on the auxiliary digraph. At the termination, we uncontract s^* and replace each edge (s^*, t) of M by the in-directed Steiner tree J_t.

We modify the algorithm by putting a restriction that the in-directed Steiner tree J_t for each vertex $t \in T$ must be formed by augmenting edges whose heads are at t. If such an in-directed Steiner tree rooted at t does not exist, then we will not add the auxiliary edge (s^*, t); alternatively, we may set cost of (s^*, t) to ∞. The modification does not violate the feasibility of the instance for the minimum cost out-branching problem. To see this, consider any feasible solution $G^* = (V, E_0 \cup E^*)$. Since any top-vertex $t \in T$ with respect to G^* has no dipath from vertices in $T - \{t\}$, augmenting edges of G^* with heads at t together with E_0 form an in-directed Steiner tree J_t^{in} with the root vertex t and the terminal set S. This means that if the given instance of the relaxed (S, T)-connectivity problem is feasible, then there must exist a solution to the minimum cost out-branching problem on the auxiliary digraph. By the proof of Theorem 2.5.9, the algorithm gives us an (S, T)-connected digraph.

The cost analysis follows from the same analysis in Theorem 2.5.9. We replace $\alpha(n)$ in the analysis by $\alpha'(n)$ which is the approximation ratio for Set Cover . However, we can get a tighter bound because, for each junction tree J_i, the in-tree J_i^{in} and the out-tree J_i^{out} have no common edges. This is because J_i^{in} and J_i^{out} have only the root vertex t_i in common, and t_i is a single vertex not a strongly connected component. Thus, the cost of the solution obtained by the algorithm is at most

$$\mathrm{c}(M) \leq \sum_{i=1}^{\ell} (\mathrm{c}(J_i^{out}) + \alpha'(n) \cdot \mathrm{c}(J_i^{in})) \leq \sum_{i=1}^{\ell} \alpha'(n)(\mathrm{c}(J_i)^{out} + \mathrm{c}(J_i)^{in}) = \alpha'(n) \sum_{i=1}^{\ell} \mathrm{c}(J_i) \leq \alpha'(n)\mathrm{opt}.$$

2.6.2 An approximation algorithm II

In this section, we give an alternative approximation algorithm for the problem. This algorithm achieves the same approximation guarantee as the algorithm in the previous section. We will now describe our algorithm.

First, we number vertices of T in topological order as $t_1, t_2, \ldots, t_{|T|}$. Precisely, we order T vertices so that there is no $t_i \to t_j$ dipath in G, for all $j < i$. It is possible to order T vertices this way because the given digraph is acyclic on T. We then iterate on each vertex in this order, say $t_1, t_2 \ldots, t_{|T|}$. For each vertex $t_i \in T$, we compute two subsets of augmenting edges $E_i^{(1)}$ and $E_i^{(2)}$ and choose the one with minimum cost, denoted by E_i. The set $E_i^{(1)}$ and $E_i^{(2)}$ are defined in the algorithm below. (We may assume that no vertex $t_i \in T$ has an $s \to t_i$ dipath from all vertices $s \in S$.)

1. **A subset of augmenting edges** $E_i^{(1)}$: This set is a set of augmenting edges obtained by computing a shortest dipath to t_i from the previous vertices (in the topological order). Precisely, we compute a shortest dipath P from $\{t_1, t_2, \ldots, t_{i-1}\}$ to t_i on the given digraph $G = (V, E_0 \cup E)$. The dipath P may contain zero-cost edges of the initial digraph; however, we only need augmenting edges in P, that is, $E_i^{(1)}$ is the set of augmenting edges in P. This set of augmenting edges gives $E_i^{(1)}$. We call this the instance $\Pi_i^{(1)}$ of the shortest dipath problem.

2. **A subset of augmenting edges** $E_i^{(2)}$: This set is computed by the greedy algorithm for Set Cover . To create the Set Cover instance, we take S as the universe. For each edge e with head at t_i, we create a subset S_e with cost $\mathrm{c}(e)$. We add an element $s \in S$ to S_e if we can gain an $s \to t_i$ dipath by adding e to the initial digraph $G_0 = (V, E_0)$, that is, $S_e = \{s \in S : G_0 + e \text{ contains an } s \to t_i \text{ dipath}\}$. Observe that S_e also contains a vertex $s \in S$ that already has an $s \to t_i$ dipath in the initial digraph $G_0 = (V, E_0)$. This set of augmenting edges gives $E_i^{(2)}$. We call this the instance $\Pi_i^{(2)}$ of Set Cover .

Analysis

Before proceeding to the analysis, we need the following lemma which characterizes the dipath between two distinct T vertices.

Lemma 2.6.5. *Consider any vertex $t_i \in T$, $1 \leq i \leq |T|$. If there exists a dipath from $\{t_1, t_2, \ldots, t_{i-1}\}$ to t_i, then a shortest such dipath P contains at most one augmenting edge. Moreover, if P has an augmenting edge e, then e has t_i as head.*

Proof. Let P be a shortest dipath from $t_j \in \{t_1, t_2, \ldots, t_{i-1}\}$ to t_i. Notice that P contains no augmenting edges with heads in $\{t_1, t_2, \ldots, t_{i-1}\}$. To see that, suppose there is an augmenting edge e having head at $t_{j'}$ for $j' < i$. Then we can remove a subpath of P with start vertex t_j

and end vertex $t_{j'}$. Since all augmenting edges have positive cost, this will give us a dipath from $\{t_1, t_2, \ldots, t_{i-1}\}$ to t_i of a cheaper cost. By similar arguments, dipath P enters t_i only once. We conclude that there is at most one augmenting edge in P. Moreover, if P has an augmenting edge e, then the head of e must be t_i, and the lemma follows. □

Corollary 2.6.6. *Consider any vertex $t_i \in T, 1 \leq i \leq |T|$. If there exists a dipath from $\{t_1, t_2, \ldots, t_{i-1}\}$ to t_i, then $E_i^{(1)}$ consists of at most one augmenting edge.*

We will now show the feasibility and approximation guarantee of the solution.

Theorem 2.6.7. *The above algorithm is an $\alpha'(n)$-approximation algorithm for the special case of relaxed (S,T)-connectivity problem where the given digraph is acyclic on T, where $\alpha'(n)$ is the best known approximation ratio for* Set Cover *.*

Proof. Let E^* denote the set of augmenting edges in an optimal solution, let $G^* = (V, E_0 \cup E^*)$, and let opt denote the cost of E^*. For $i = 1, 2, \ldots, |T|$, let E_i^* denote the subset of edges of E^* that have heads at $t_i \in T$. Since augmenting edges are restricted to have heads in T, the subsets $E_1^*, E_2^*, \ldots, E_{|T|}^*$ form a partition of E^*. Let $T_i = \{t_1, t_2, \ldots, t_i\}$. Let $G_i = (V, E_0 \cup E_1 \cup E_2 \ldots \cup E_i)$, where E_j is the set of edges chosen by the algorithm for $j = 1, 2, \ldots, i$. In fact, G_i is the digraph obtained by the algorithm at the iteration i. Recall that the algorithm chooses $E_i = E_i^{(1)}$ or $E_i = E_i^{(2)}$ depending on which set has minimum cost, that is, $E_i = \operatorname{argmin}\left\{\mathrm{c}(E_i^{(1)}), \mathrm{c}(E_i^{(2)})\right\}$.

We will proceed by induction on i for $i = 1, 2, \ldots, |T|$ to show the followings:

- (i) At least one of the two subproblems $\Pi_i^{(1)}$ and $\Pi_i^{(2)}$ (See Section 2.6.2.) is feasible.
- (ii) G_i is (S, T_i)-connected.
- (iii) $\mathrm{c}(E_j) \leq \alpha'(n)\mathrm{c}(E_j^*)$.

We recall that $\Pi_i^{(1)}$ is the instance of the shortest $T_{i-1} \to t_i$ dipath problem, and $\Pi_i^{(2)}$ is the instance of Set Cover . Note that the number of elements in the instance $\Pi_i^{(2)}$ of Set Cover is $|S| = O(n)$, and the greedy algorithm approximates Set Cover within a factor of $\alpha'(n)$.

Before proceeding, we note that if the instance $\Pi_i^{(2)}$ of Set Cover is feasible, then adding $E_i^{(2)}$ to the digraph makes it (S, t_i)-connected. This is because of the construction of the instance of $\Pi_i^{(2)}$.

Base case i=1: The base case follows by the way we order T vertices. Since t_1 is the first vertex in the order, there is no $S \to t_1$ dipath containing vertices of $T - \{t_1\}$. This means that any set of augmenting edges that makes the digraph $(S, \{t_1\})$-connected must have heads at t_1. Thus, if the instance of the (S,T)-connectivity problem is feasible, then so is the instance $\Pi_1^{(2)}$ of Set Cover ,

proving (i) and (ii). But, there is no solution for $E_1^{(1)}$ because t_1 is the first vertex in the ordering. So, $E_1 = E_1^{(2)}$, and G_1 is $(S, \{t_1\})$-connected. Since there is no $t_j \to t_i$ dipath in $G = (V, E_0 \cup E)$ for all $j > i$, we conclude that augmenting edges in E_1^* form a feasible solution to Set Cover . It follows that $\mathrm{c}(E_1) \leq \alpha'(n)\mathrm{c}(E_1^*)$, proving (iii).

Inductive step $i > 1$**:** We assume that the induction hypothesis holds for some $1 \leq i < |T|$. Note that (ii) follows from (i). To see that, first, if the instance $\Pi_{i+1}^{(2)}$ of Set Cover is feasible, then adding $E_{i+1}^{(2)}$ to the digraph makes it (S, T_{i+1})-connected. Second, if there is a $t_j \to t_{i+1}$ dipath for some $1 \leq j \leq i$, then by the induction hypothesis, there is a dipath to t_{i+1} from every vertex $s \in S$ of the form $s \to \ldots \to t_j \to \ldots \to t_{i+1}$. Hence, it suffices to prove (i) and (iii). Consider two cases.

- **Case 1: There is a** $T_i \to t_{i+1}$ **dipath in** G^***.** It follows from the assumption that $\Pi_{i+1}^{(1)}$ is feasible, proving (i). Let P^* denote a shortest $T_i \to t_{i+1}$ dipath in G^*, and let P denote a shortest $T_i \to t_{i+1}$ dipath obtained by solving the instance $\Pi_{i+1}^{(1)}$. By Lemma 2.6.5, there is at most one augmenting edge in P^*, and if P^* has an augmenting edge e, e must have head at t_{i+1} which means that $e \in E_{i+1}^*$. By the minimality of $E_{i+1}^{(1)}$ and E_{i+1}, we have $\mathrm{c}(E_{i+1}) \leq \mathrm{c}(E_{i+1}^{(1)}) \leq \mathrm{c}(E_{i+1}^*) \leq \alpha'(n)\mathrm{c}(E_{i+1}^*)$, proving (iii).
- **Case 2: There is no** $T_i \to t_{i+1}$ **dipath in** G^***.** By the topological ordering, there is no $t_j \to t_{i+1}$ dipath for $j > i+1$. So, every $S \to t_{i+1}$ dipath in G^* contains no T vertices other than t_{i+1}. Hence, for all $s \in S$, all augmenting edges of an $s \to t_{i+1}$ dipath have heads at t_{i+1}. This implies that the instance $\Pi_{i+1}^{(2)}$ of Set Cover is feasible, and (i) holds. This also implies that the edges of E_{i+1}^* form a feasible solution to $\Pi_{i+1}^{(2)}$. By the minimality of E_{i+1}, we have $\mathrm{c}(E_{i+1}) \leq \mathrm{c}(E_{i+1}^{(2)}) \leq \alpha'(n)\mathrm{c}(E_{i+1}^*)$, proving (iii).

As a consequence of the above claim, we have that $G_{|T|}$ is (S, T)-connected and

$$\sum_{i=1}^{|T|} \mathrm{c}(E_i) \leq \sum_{i=1}^{|T|} O(\alpha'(n)\mathrm{c}(E_i^*) \leq O(\alpha'(n)\mathrm{opt}.$$

Therefore, our algorithm is $\alpha'(n)$-approximation algorithm, where $\alpha'(n)$ is the approximation ratio for approximating Set Cover .

□

Chapter 3

An approximation algorithm for the k-(S,T)-connectivity problem

3.1 Introduction

In this chapter, we discuss the k-(S,T)-connectivity problem. This problem is a generalization of the (S,T)-connectivity problem discussed in the previous chapter. In the (S,T)-connectivity problem, we are asked to find a subgraph that has a dipath from every vertex of S to every vertex of T. In the k-(S,T)-connectivity problem, we are asked to find a subgraph that has k edge-disjoint dipaths from each vertex $s \in S$ to each vertex $t \in T$.

We study the problem in the standard setting, that is, all augmenting edges have tails in S and heads in T. This problem also generalizes both the minimum cost directed k-edge connected spanning subgraph problem (k-ECSS) and the minimum cost directed k-vertex connected spanning subgraph problem (k-VCSS). Extending the algorithm of Fakcharoenphol and Laekhanukit for k-VCSS in [24], we design a polylogarithmic-approximation algorithm for this problem.

3.1.1 Organization

The organization of this chapter is as follows. Section 3.2 is a preliminary section in which we introduce some definitions and notations. In Section 3.3, we present the hardness result of the k-(S,T)-connectivity problem. In Section 3.4, we give some introductory discussion on the k-(S,T)-connectivity problem. We also introduce the (S,T)-connectivity augmentation problem. In Section 3.9, we move our focus to the (S,T)-connectivity augmentation problem.

3.2 Preliminaries

In the minimum cost k-(S,T)-connectivity problem, we are given a digraph $G=(V, E_0 \cup E)$, two sets of vertices S and T, and a positive integer k. We call $G_0=(V,E_0)$ the *initial digraph* and call edges in E *augmenting edges*. We also have non-negative cost assigned to augmenting edges. We may assume that the edges in E_0 have zero-cost while the edges in E have positive cost. The set of vertices $S \cup T$ might not contain all vertices. We call vertices that are in $V-(S\cup T)$ *optional vertices*. We say that a digraph is k-(S,T)-*connected* if there are k edge-disjoint $s \to t$ dipaths connecting every pair of vertices $s \in S$ and $t \in T$. The (S,T)-*connectivity* of the digraph G is the minimum number ρ such that G is ρ-(S,T)-connected. The goal in this problems is to find a minimum cost subset of edges $E' \subseteq E$ so that the digraph $G'=(V, E_0 \cup E')$ is k-(S,T)-connected.

We assume that all augmenting edges have tails in S and have heads in T. Recall that in the previous chapter we studied several different versions of the (S,T)-connectivity problem, and in the standard version, the augmenting edges had tails in S and heads in T. In this chapter, we study only this version.

Also, we may assume that S and T are disjoint by extending the vertex-splitting technique presented in the previous chapter. (See Proposition 2.2.1.) Recall that we split every vertex $v \in S \cap T$ into v^+ and v^-, direct edges entering v to v^-, direct edges leaving v to v^+, and add an auxiliary edge (v^-, v^+) with zero-cost. The reduction for the k-(S,T)-connectivity problem is the same as that of the (S,T)-connectivity problem except that we add k parallel edges from v^- to v^+ instead of one auxiliary edge. Note that if parallel edges are not allowed, we can subdivide each edge (v^-, v^+) by a dipath $v^- \to u_i \to v^+$, for $i=1,2,\ldots,k$. See Figure 3.1.

Proposition 3.2.1. *There is a reduction from instances of the k-(S,T)-connectivity problem where $S \cap T \neq \emptyset$ to instances such that $S \cap T = \emptyset$ that preserves the feasibility and the cost of the solution.*

Throughout this chapter, we use n and m to denote the number of vertices and the number of edges, respectively. We denote the set of edges in an optimal solution by E^* and denote its cost by opt. Let F be any set of edges. For any set of vertices $U \subseteq V$, we use $\delta^{\mathrm{out}}_F(U)$ to denote the set of edges of F leaving U and use $d^{\mathrm{out}}_F(U)$ to denote its number, that is,

$$\delta^{\mathrm{out}}_F(U) = \{e \in F : \text{ the tail of } e \text{ is in } U, \text{ and the head of } e \text{ is not in } U\} \text{ and } d^{\mathrm{out}}_F(U) = |\delta^{\mathrm{out}}_F(U)|.$$

If it is clear in the context, then we will omit the subscript F and write them as $\delta^{\mathrm{out}}(U)$ and $d^{\mathrm{out}}(U)$. For any digraph $G=(V,E)$, we use the notation $G+F=(V, E\cup F)$. Also, let $\boldsymbol{x}$ be a vector of variables of an LP. Then we will use the notation $\boldsymbol{x}(F)=\sum_{e\in F} x_e$.

3.3 The hardness of the k-(S,T)-connectivity problem

In this section, we give the hardness result of the k-(S,T)-connectivity problem. It is clear that this problem generalizes the minimum cost directed k-edge connected spanning subgraph problem

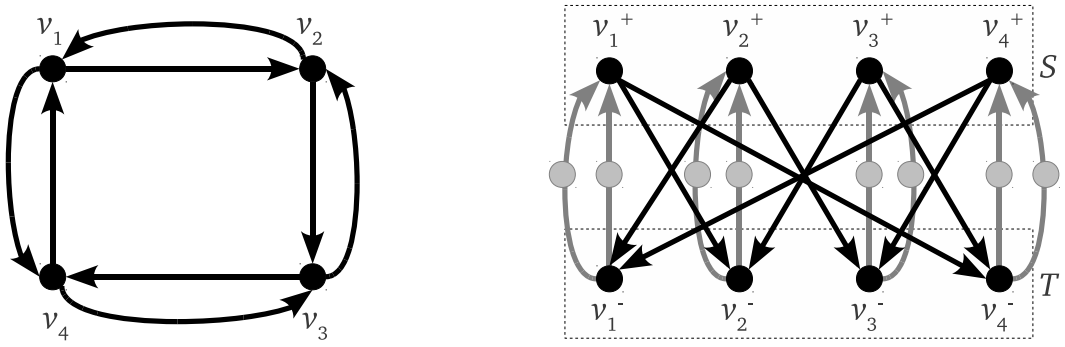

Figure 3.1: The figure shows the reduction from the instance of the k-(S,T)-connectivity problem where $S \cap T \neq \emptyset$ to the instance such that $S \cap T = \emptyset$. The left figure is the original instance where $S = T = V$, and the right figure is the transformed instance where $S \cap T = \emptyset$. The original instance is, in fact, the instance of the minimum cost 2-edge connected subgraph problem. In this figure, the black lines denote positive cost edges, and the grey lines denote zero-cost edges. The grey circles denote the vertices added to subdivide the auxiliary edges.

(k-ECSS). In fact, we may think of each instance of k-ECSS as an instance of the k-(S,T)-connectivity problem where $S = T = V$. See Figure 3.1.

The k-(S,T)-connectivity problem also generalizes the minimum cost directed k-vertex connected spanning subgraph problem (k-VCSS). The reduction is the same as that of the splitting technique except that we add only one auxiliary edge (v^-, v^+) for each vertex $v \in V$. Notice that there is a one-to-one correspondence between a vertex v of the original digraph and an auxiliary edge (v^-, v^+) of the transformed one. This gives us a one-to-one mapping between dipaths of the two instances. Consider any k edge-disjoint $s^+ \to t^-$ dipaths $P_1, P_2, \ldots, P_k$ between any pair of vertices $s^+ \in S$ and $t^- \in T$ of a feasible solution to the k-(S,T)-connectivity instance. For $i = 1, 2, \ldots, k$, let $\hat{P}_i$ denote the corresponding dipath of P_i in the original instance of k-VCSS . Clearly, $P_1, P_2, \ldots, P_k$ have no auxiliary edges in common which implies that $\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_k$ also have no vertices in common except s and t. In other words, s and t are connected by k internally disjoint $s \to t$ dipaths. Conversely, consider any k internally disjoint $s \to t$ dipaths $\hat{P}_1, \hat{P}_2, \ldots, \hat{P}_k$ of the feasible solution to k-VCSS . Then the corresponding dipaths $P_1, P_2, \ldots, P_k$ must be edge disjoint. This is because, for any two dipaths $\hat{P}_i$ and $\hat{P}_j$, $1 \leq i \neq j \leq k$, $\hat{P}_i$ and $\hat{P}_j$ have no edges and no internal vertices in common which implies that the corresponding dipaths P_i and P_j also have no edges in common including the auxiliary ones. Thus, the solution to the instance of k-VCSS is feasible if and only if the corresponding solution to the instance of the k-(S,T)-connectivity problem is feasible.

The k-VCSS problem in undirected graphs is known to be APX-hard [20, 41]. The same hardness applies to the problem in directed graphs. Lando and Nutov showed in [58] that if there

is a ρ-approximation algorithm for k-VCSS in undirected graphs, then there is a ρ-approximation algorithm for k-VCSS in directed graphs. This implies that the k-VCSS problem in directed graph is also APX-hard.

The hardness results of k-VCSS and the k-(S,T)-connectivity problem are as follows.

Theorem 3.3.1 ([20, 41] and [58])**.** *The minimum cost directed k-vertex connected spanning subgraph problem is* APX*-hard.*

Theorem 3.3.2. *The k-(S,T)-connectivity problem is at least as hard as the minimum cost directed k-vertex connected spanning subgraph problem.*

3.4 Tools for the approximation algorithm for the k-(S,T)-connectivity problem

In the previous section, we showed that the k-(S,T)-connectivity problem generalizes both k-ECSS and k-VCSS . Although the three are in different settings, they share some common properties. For example, different versions of Menger's Theorem hold for each of these problems.

Theorem 3.4.1 (Menger's Theorem)**.** *Consider a digraph G and pair of vertices s,t of G. Then the following holds.*

- *There are k edge-disjoint $s \to t$ dipaths if and only if there is no set of less than k edges whose removal disconnects s and t.*
- *There are k internally (vertex) disjoint $s \to t$ dipaths if and only if there is no set of less than k vertices of $V - \{s,t\}$ whose removal disconnects s and t.*

Clearly, the edge-version of Menger's Theorem holds for k-ECSS while the vertex-version holds for k-VCSS . In the k-(S,T)-connectivity problem, the edge-version of Menger's Theorem holds for every pair of vertices $s \in S$ and $t \in T$, that is, the solution $G' = (V, E_0 \cup E')$ is feasible to this problem if and only if there is no set of less than k edges whose removal disconnects a pair of vertices $s \in S$ and $t \in T$.

Applying Menger's Theorem, we have the following constraints of the k-(S,T)-connectivity problem, where E' is the set of edges chosen by the algorithm. The constraints also give us a linear programming (LP) relaxation for the k-(S,T)-connectivity problem as described in Table 3.4.

$$d^{\text{out}}_{E_0 \cup E'}(U) \geq k \text{ for all } U \subseteq V, U \cap S \neq \emptyset \text{ and } U \cap T \neq T$$

A common framework for solving connectivity problems, e.g., k-ECSS and k-VCSS , is the connectivity augmentation framework. Under this framework, we start with a digraph of zero

$$\mathrm{LP}(k)\begin{cases} z_k^* = \min & \sum_{e\in E} \mathrm{c}(e)\cdot x_e & \\ \text{s.t.} & \boldsymbol{x}(\delta^{\mathrm{out}}_{E-E_0}(U)) + d^{\mathrm{out}}_{E_0}(U) \geq k & \forall U \subseteq V, U\cap S\neq\emptyset, U\cap T\neq T \\ & 0\leq x_e\leq 1 & \forall e\in E-E_0 \end{cases}$$

Table 3.1: The LP relaxation of the k-(S,T)-connectivity problem

(edge or vertex) connectivity and then add some edges to increase the connectivity of the digraph to $1, 2, \ldots, k$. The technique was used in [16, 24, 42, 56, 62, 63, 67].

The connectivity augmentation framework applies to the k-(S,T)-connectivity problem. We first start by the initial digraph $G_0 = (V, E_0)$. (Assume that G_0 has zero (S,T)-connectivity.) Then we find a subset of edges $E_1 \subseteq E$ so that $G_1 = G + E_1$ is 1-(S,T)-connected. Generally, at the i-th iteration, we start from the initial digraph G_{i-1}, and we find a subset of edges $E_i \subseteq E$ such that $G_i = G_{i-1} + E_i$ is i-(S,T)-connected. We repeat the process until the digraph G_k is k-(S,T)-connected.

In fact, we focus on solving instances of the (S,T)-*connectivity augmentation* problem. Consider the optimal solution $\boldsymbol{x}^*$ to the LP for the k-(S,T)-connectivity problem. Observe that if the initial digraph is ℓ-(S,T)-connected, then $\left(\frac{1}{k-\ell}\right)\boldsymbol{x}$ is feasible to $\mathrm{LP}_{aug}(\ell)$. (See Lemma 3.4.2.) Hence, we can compare the cost of the optimal solution of the (S,T)-connectivity augmentation problem to the optimal value of $\mathrm{LP}(k)$ by multiplying by $\frac{1}{k-\ell}$. This technique is called the ***LP-scaling technique***.

Using LP-scaling technique, we can show that the connectivity augmentation framework gives us an $O(\alpha(n)\log k)$-approximation algorithm for the k-(S,T)-connectivity problem, where $\alpha(n)$ is the approximation ratio for approximating the (S,T)-connectivity augmentation problem. In other words, we only have to pay an additional factor of $O(\log k)$ to increase the (S,T)-connectivity of the initial digraph from 0 to k.

To see the claim, consider the following LPs defined on the same given digraph $G = (V, E_0 \cup E)$.

- The LP relaxation $\mathrm{LP}(k)$ of the k-(S,T)-connectivity problem in Table 3.4, where E_0 is the set of edges of the initial digraph $G_0 = (V, E_0)$.
- The LP relaxation $\mathrm{LP}_{aug}(\ell)$ of the (S,T)-connectivity augmentation problem in Table 3.2, where E_ℓ is the set of edges of the initial digraph $G_\ell = (V, E_\ell)$ which is ℓ-(S,T)-connected.

Before proceeding to prove the main claim, we need the following lemma which shows that the cost of the optimal *fractional solution* to $\mathrm{LP}_{aug}(\ell)$ is at most $1/(k-\ell)$ of that of $\mathrm{LP}(k)$. The proof proceeds by the LP-scaling technique. Note that similar lemmas were established in [16, 24, 42, 56, 62, 63, 67].

$$\mathrm{LP}_{aug}(\ell)\begin{cases} z^*_{aug} = \min & \displaystyle\sum_{e\in E-E_\ell} \mathrm{c}(e)\cdot x_e & \\ \quad \text{s.t} & \boldsymbol{x}(\delta^{\mathrm{out}}_{E-E_\ell}(U)) \geq 1 & \forall U \subseteq V, U\cap S\neq\emptyset, U\cap T\neq T, d^{\mathrm{out}}_{E_0}(U)=\ell \\ & 0\leq x_e\leq 1 & \forall e\in E-E_\ell \end{cases}$$

Table 3.2: The LP relaxation of the (S,T)-connectivity augmentation problem

Lemma 3.4.2. *Given the digraph $G=(V,E_0\cup E)$ and the initial digraph $G=(V,E_\ell)$ which is ℓ-(S,T)-connected, where $\ell<k$, the optimal value of $\mathrm{LP}_{aug}(\ell)$ is at most $1/(k-\ell)$ times the optimal value of $\mathrm{LP}(k)$, that is, $z^*_{aug}\leq z^*_k/(k-\ell)$, where z^*_{aug} and z^*_k denote the optimal value of $\mathrm{LP}_{aug}(\ell)$ and LP_k, respectively.*

Proof. Let $\boldsymbol{x}^*$ be an optimal solution to $\mathrm{LP}(k)$, and let E^* denote the corresponding support, that is, $E^*=\{e\in E : x^*_e>0\}$.

We construct a solution $\boldsymbol{x}'$ to $\mathrm{LP}_{aug}(\ell)$ by assigning $x'_e=1/(k-\ell)$ to all edges $e\in E^*-E_\ell$ and $x'_e=0$ otherwise. We claim that $\boldsymbol{x}'$ is feasible to $\mathrm{LP}_{aug}(\ell)$. To see this, consider any set of vertices $U\subseteq V$ appearing in the constraints of $\mathrm{LP}(\ell)$. Since $\boldsymbol{x}^*$ satisfies $\mathrm{LP}(k)$, we have $\boldsymbol{x}^*(\delta^{\mathrm{out}}_{E^*}(U))+d^{\mathrm{out}}_{E_0}(U)\geq k$. Hence,

$$\boldsymbol{x}^*(\delta^{\mathrm{out}}_{E^*-E_\ell}(U))+d^{\mathrm{out}}_{E_\ell}(U)\geq \boldsymbol{x}^*(\delta^{\mathrm{out}}_{E^*}(U))+d^{\mathrm{out}}_{E_0}(U)\geq k$$

Moreover, we have $d^{\mathrm{out}}_{E_\ell}(U)=\ell$ by the constraints of $\mathrm{LP}_{aug}(\ell)$. This implies that $\boldsymbol{x}^*(\delta^{\mathrm{out}}_{E^*-E_\ell}(U))\geq k-\ell$, and thus, $\boldsymbol{x}'(\delta^{\mathrm{out}}_{E^*-E_\ell}(U))\geq 1$. Hence, $\boldsymbol{x}'$ is feasible to LP_{aug}. Clearly, the cost of x' is at most $1/(k-\ell)$ of that of $\boldsymbol{x}^*$, proving the lemma. □

We are now ready to prove the main claim.

Theorem 3.4.3. *Suppose there is an approximation algorithm for the (S,T)-connectivity augmentation problem that achieves an approximation guarantee of $\alpha(n)$ with respect to the standard LP relaxation, that is, the cost of the solution found by the algorithm is at most $\alpha(n)\cdot\mathrm{opt}$. Then there is an $O(\alpha(n)\log k)$-approximation algorithm for the k-(S,T)-connectivity problem.*

Proof. The proof follows from Lemma 3.4.2. We solve the given instance of the k-(S,T)-connectivity problem by iteratively increasing the (S,T)-connectivity of the digraph by one via the $\alpha(n)$-approximation algorithm. Note that this algorithm finds a solution of cost at most $\alpha(n)\cdot z^*_{aug}\leq \alpha(n)\cdot z^*_k/(k-\ell)$. Hence, the resulting set of augmenting edges has cost at most

$$\frac{z^*_k}{k}\alpha(n)+\frac{z^*_k}{k-1}\alpha(n)+\ldots+\frac{z^*_k}{1}\alpha(n)\leq\left(\frac{1}{k}+\frac{1}{k-1}+\ldots+1\right)\alpha(n)z^*_k=O(\alpha(n)\log k)z^*_k$$

□

Remark: In the rest of this chapter, we will discuss the (S,T)-connectivity augmentation problem. In fact, we assume that we are at the ℓ-th iteration of the connectivity augmentation framework, and hence, the current digraph is ℓ-(S,T)-connected. Thus, we make the following assumption.

Assumption 3.4.4. *The initial digraph $G_\ell = (V, E_\ell)$ is ℓ-(S,T)-connected.*

In Chapter 3.9, we will present two approximation algorithms with the approximation guarantee of $O(\log n)$. Thus, we get the following result for the k-(S,T)-connectivity problem.

Theorem 3.4.5 (Main Theorem)**.** *There exists an $O(\log n \log k)$-approximation algorithm for the k-(S,T)-connectivity problem.*

Another problem closely related to the k-(S,T)-connectivity problem is the *rooted k-connection problem*. In the rooted k-connection problem, we are given a digraph $G = (V, E_0 \cup E)$, a root vertex r, a set of terminals $T \subseteq V - \{r\}$, and a positive integer k; in addition, all augmenting edges in E have heads in T. Similar to the k-(S,T)-connectivity problem, the edges of the initial digraph $G_0 = (V, E_0)$ have zero-cost while the augmenting edges of E have positive cost. We say that a digraph is *k-connected to T from r* or *k-(r,T)-connected* if there are k edge-disjoint dipaths from r to each vertex of T. The goal in the rooted k-connection problem is to find a set of augmenting edges $E' \subseteq E$ with minimum cost so that $G' = (V, E_0 \cup E')$ is k-connected to T from r. In fact, the rooted k-connection problem is the special case of the k-(S,T)-connectivity problem, where $S = \{r\}$, and all augmenting edges are restricted to have heads in T. The rooted problem has been showed to be polynomial-time solvable by Frank [37]. In fact, Frank [37] showed that the standard linear programming (LP) relaxation of this problem has an integral optimal solution. Similar to the k-(S,T)-connectivity problem, an important special case of the rooted k-connection problem is the *rooted connectivity augmentation problem*. This is the special case where the initial digraph $G_0 = (V, E_0)$ is ℓ-(r,T)-connected and $k = \ell + 1$. Under the condition that all augmenting edges have heads in T, the rooted connectivity augmentation problem can be solved in $O(n^2m + n \cdot t(m,n))$ time, where $t(m,n)$ is the time for computing a maximum s,t-flow.

Observe that every k-(S,T)-connected digraph is rooted k-connected to T from every vertex $s \in S$. Thus, we may solve the (S,T)-k-connectivity problem by taking T as the terminal set, and applying the algorithm for the rooted k-connection problem rooted at each vertex $s \in S$. This gives us the trivial $O(n)$-approximation algorithm for the k-(S,T)-connectivity problem. Our algorithms, which will be presented in the later sections, follow the same idea; however, with some care, we gain a better bound from the rooted subroutine. We will now state the results on the rooted k-connection problem and the rooted connectivity augmentation problem.

Theorem 3.4.6 (Frank 2009 [37])**.** *The rooted k-connection problem where all augmenting edges have heads in T is polynomial-time solvable. Moreover, the cost of the optimal solution to this problem is equal to the optimal value of the standard LP relaxation.*

Theorem 3.4.7 (Frank 1999 [35])**.** *The rooted connectivity augmentation problem where all augmenting edges have heads in T is solvable in $O(n^2m + n \cdot t(m,n))$ time, where $t(m,n)$ is the time*

for computing a maximum s,t-flow. Moreover, the cost of the optimal solution to this problem is equal to the optimal value of the standard LP relaxation.

Note that the standard LP relaxations of the rooted k-connection problem and the rooted connectivity augmentation problem are the same as that of the k-(S,T)-connectivity problem and the (S,T)-connectivity augmentation problem, respectively. The LPs are presented in Table 3.4 and Table 3.2.

3.5 Preliminaries on ℓ-(S,T)-connected digraphs

Consider the initial digraph $G_\ell = (V, E_\ell)$ which is ℓ-(S,T)-connected. A *deficient set* is a set of vertices $U \subseteq V$ such that $U \cap S \neq \emptyset$, $U \cap T \neq T$, and $d^{\text{out}}(U) < \ell + 1$. Observe that U separates some pair of vertices $s \in S$ and $t \in T$, that is, $s \in U$ and $t \in V - U$. Hence, removing the edges of $\delta^{\text{out}}(U)$ disconnects vertices s and t. Since $d^{\text{out}}(U) < \ell + 1$, the existence of the deficient set U implies that the digraph has (S,T)-connectivity less than $\ell + 1$. Observe that every deficient set U has $d^{\text{out}}(U) = \ell$ because the initial digraph is ℓ-(S,T)-connected by Assumption 3.4.4.

The next lemma gives a basic property of deficient sets which follows from the submodularity of $d^{\text{out}}(.)$.

Lemma 3.5.1 (Uncrossing Lemma)**.** *Let U and W be two deficient sets such that $(U \cap W) \cap S \neq \emptyset$ and $(U \cup W) \cap T \neq T$. Then both $U \cap W$ and $U \cup W$ are deficient sets.*

Proof. First, it is well-known that the function $d^{\text{out}}(.)$ is submodular, that is,

$$d^{\text{out}}(U) + d^{\text{out}}(W) \geq d^{\text{out}}(U \cap W) + d^{\text{out}}(U \cup W).$$

Since U and W are deficient sets, we have $d^{\text{out}}(U) = d^{\text{out}}(W) = \ell$. Moreover, by the hypothesis of the lemma, $U \cap W \cap S \neq \emptyset$ and $T - (U \cup W) \neq \emptyset$. This implies that $d^{\text{out}}(U \cap W) \geq \ell$ and $d^{\text{out}}(U \cup W) \geq \ell$ because the initial digraph G_ℓ is ℓ-(S,T)-connected. It then follows by the submodularity of $d^{\text{out}}(.)$ that $d^{\text{out}}(U \cap W) = d^{\text{out}}(U \cup W) = \ell$. Therefore, both $U \cap W$ and $U \cup W$ are deficient sets.

□

We call an inclusionwise minimal deficient set C, a *core*. By the way we define core, it is clear that any deficient set must contain at least one core; possibly, the set may be a core itself. The *halo-family* of the core C, denoted by $\text{Halo}(C)$, is the family of deficient sets that contains C but contains no other cores, that is,

$$\text{Halo}(C) = \{U : U \text{ is a deficient set }, C \subseteq U, U \text{ contains no other cores}\}.$$

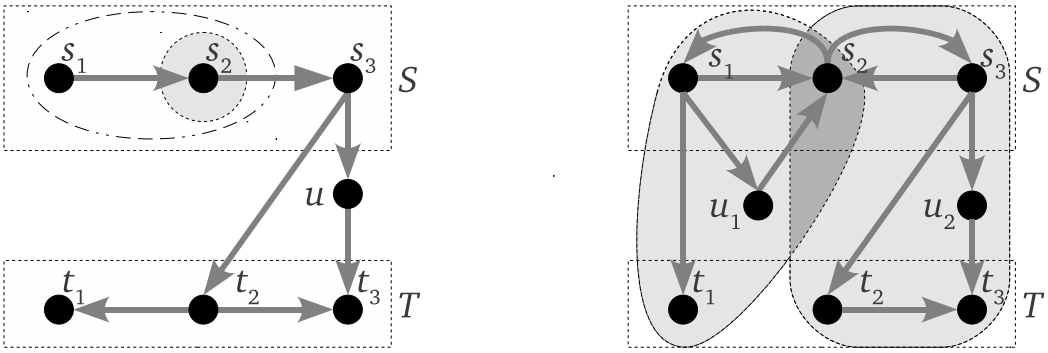

Figure 3.2: The figure shows two instances of the k-(S,T)-connectivity problem, where the initial digraph is 1-(S,T)-connected. The cores in the left instance are $\{s_1\}$, $\{s_2\}$, and $\{s_3\}$. The deficient set $\{s_1, s_2\}$ is not a core because it properly contains $\{s_1\}$ and $\{s_2\}$. The cores in the right instance are $\{s_1, s_2, u_1, t_1\}$ and $\{s_2, s_3, u_2, t_2, t_3\}$. These two cores have a common vertex s_2.

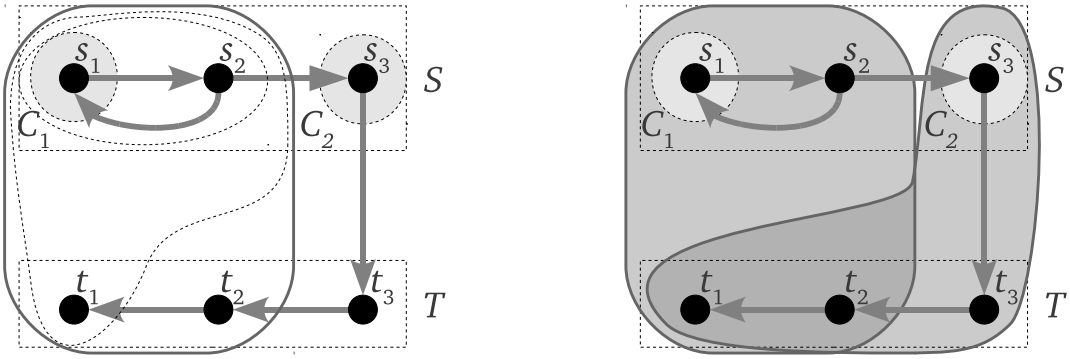

Figure 3.3: The figure shows the instance of the k-(S,T)-connectivity problem, where the initial digraph is 1-(S,T)-connected. The grey circles denote the cores. The white circles denote the deficient sets which are in one of the halo-families. The circles with thick lines denote halo-sets. The right figure shows that two halo-sets can intersect. The cores in the left figure are $C_1 = \{s_1\}$ and $C_2 = \{s_3\}$. The deficient sets in the halo-family of C_1 are $\{s_1\}$, $\{s_1, s_2\}$, $\{s_1, s_2, t_1\}$ and $\{s_1, s_2, t_1, t_2\}$. The deficient sets in the halo-family of C_2 are $\{s_3\}$, $\{s_3, t_3\}$, $\{s_3, t_2, t_3\}$, $\{s_3, t_1\}$ and $\{s_3, t_1, t_2\}$. Note that a deficient set $\{s_1, s_2, s_3\}$ is not in any of the halo-families because it contains two cores.

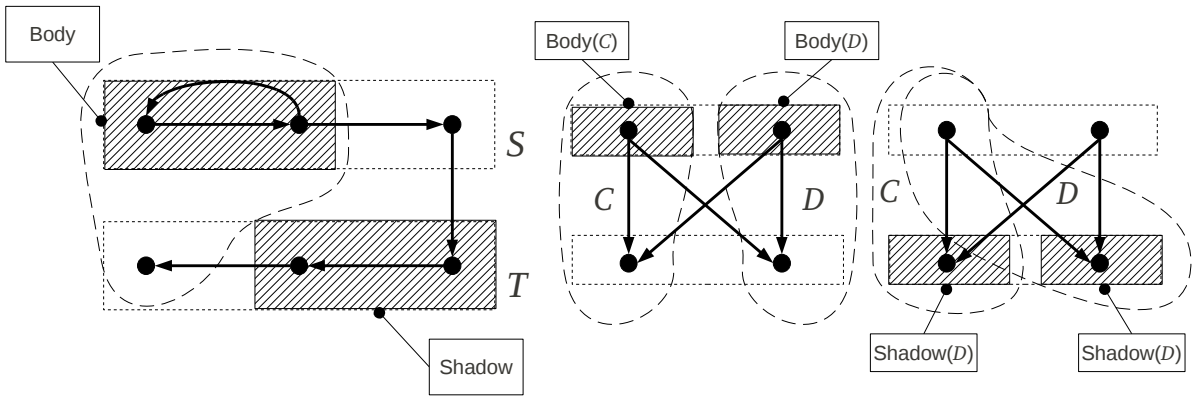

Figure 3.4: The left figure illustrates the body and the shadow of the deficient set in the circle. The right figure illustrates the statement of Lemma 3.5.2 (Disjointness Property).

The *halo-set* of the core C, denoted by $\mathrm{H}(C)$, is the union of all deficient sets in the halo-family of C, that is, $\mathrm{H}(C) = \bigcup\{U : U \in \mathrm{Halo}(C)\}$. The notion of core and halo-family are first introduced in [66] and [55], respectively.

We say that an edge $e = (s, t)$ *covers* a deficient set U if e has its tail in U and its head in $V - U$, that is, $s \in U$ and $t \in V - U$. Similarly, we say that a set of edges F *covers* $\mathrm{Halo}(C)$ if every deficient set in $\mathrm{Halo}(C)$ is covered by some edge $e \in F$.

For a deficient set U, we define the *body of* U to be $\mathrm{Body}(U) = U \cap S$, and we define the *shadow of* U to be $\mathrm{Shadow}(U) = T - U$.

Lemma 3.5.2 (Disjointness Property)**.** *Let C, D be two distinct cores. Let U be a deficient set in $\mathrm{Halo}(C)$, and let W be a deficient set in $\in \mathrm{Halo}(D)$. Then either*

- $\mathrm{Body}(U)$ *and* $\mathrm{Body}(W)$ *are disjoint, or*
- $\mathrm{Shadow}(U)$ *and* $\mathrm{Shadow}(W)$ *are disjoint.*

Proof. If $\mathrm{Body}(U)$ and $\mathrm{Body}(W)$ are disjoint, then we are done. Suppose that $\mathrm{Body}(U)$ and $\mathrm{Body}(W)$ intersect. By the way of contradiction, suppose that $\mathrm{Shadow}(U)$ and $\mathrm{Shadow}(W)$ also intersect. Then we have

$$\mathrm{Body}(U) \cap \mathrm{Body}(W) = (U \cap S) \cap (W \cap S) = (U \cap W) \cap S \neq \emptyset$$
$$\mathrm{Shadow}(U) \cap \mathrm{Shadow}(W) = (T - U) \cap (T - W) = T - (U \cup W) \neq \emptyset.$$

But, then by Lemma 3.5.1, $U \cap W$ is a deficient set, and thus, it must contain a core. So, we have a contradiction because C is the unique core of U, D is the unique core of W, and $C \neq D$. □

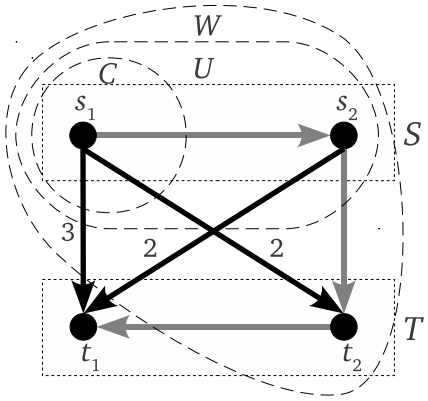

Figure 3.5: This figure shows the example of the instance of the k-(S,T)-connectivity problem, where $k = 2$, and the initial digraph is 1-(S,T)-connected. The grey lines denote the edges of the initial digraph. The black lines denote the augmenting edges. The dash lines denote the deficient sets.

At this point, we have all the definitions needed to describe the main algorithms. We illustrate our algorithm by applying it to a small example.

Consider the following instance of the k-(S,T)-connectivity problem, where $k = 2$, on the input digraph $G = (V, E_0 \cup E)$ defined as follows. See Figure 3.5.

$V = \{s_1, s_2, t_1, t_2\},\ S = \{s_1, s_2\},\ T = \{t_1, t_2\}$
$E_0 = \{(s_1, s_2), (s_2, s_1), (s_2, t_2), (t_2, t_1), (t_1, t_2)\},\ E = \{(s_1, t_1), (s_1, t_2), (s_2, t_1)\}$, and
$\mathrm{c}(s_1, t_1) = 2, \mathrm{c}(s_1, t_1) = 3, \mathrm{c}(s_2, t_1) = 2$

Notice that the initial digraph $G_0 = (V, E_0)$ is 1-(S,T)-connected; thus, this is an instance of the (S,T)-connectivity augmentation problem. One may verify that all the deficient sets in the digraph G_0 are

$$C = \{s_1\}, U = \{s_1, s_2\},\ \text{and}\ W = \{s_1, s_2, t_2\}.$$

The unique core in this digraph is C because it contains no other deficient sets, and $\mathrm{Halo}(C) = \{C, U, W\}$ because $C \subseteq U \subseteq W$ and all of them contain a unique core C; thus, $\mathrm{H}(C) = C \cup U \cup W = W$.

A simple method to add one more dipath from each vertex of S to each vertex of T is to add an augmenting edge out of each deficient set found by the algorithm. For example, once we found C, we then add an augmenting edge (s_1, t_2) to cover C. Then the new digraph has W as a deficient set; we then cover W by add an augmenting edge (s_2, t_1). The resulting digraph $G_0 + \{(s_1, t_2), (s_2, t_1)\}$ is 2-(S,T)-connected; however, its cost is not optimal because we can add the edge (s_1, t_1) to cover all deficient sets. In this example, if we cover the deficient set one-by-one

greedily, then we end up paying the cost of 4 while the cost of the optimal solution is 3. In general, we may set $c(s_1, t_2) = c(s_2, t_1) = x$ and $c(s_1, t_1) = x - 1$, and thus, the solution obtained by the greedy algorithm has cost almost twice the optimal cost. The power of Frank's algorithm from Theorem 3.4.7 is that the algorithm can optimally cover all deficient sets in the same halo-family. In fact, it will cover all deficient sets that contains a particular vertex. So, when there are two or more cores, the algorithm will pay a suboptimal cost, similarly to the naive algorithm presented above.

The main idea of our algorithms is to make use of the Frank's algorithm to cover a lot of deficient sets at the same time. In fact, we want to cover all the deficient sets that contain the same core C, for every core C in the digraph. This ensures that we will cover all the deficient sets, and thus, the (S, T)-connectivity of the digraph will increase by at least one. However, when there are two or more cores in the digraph, there might be some deficient sets that will be covered several times by Frank's algorithm. For example, suppose there is a deficient set U that contains two cores C and D that share no common vertex. Then we have to apply Frank's algorithm twice because C and D will not be considered in the same instance of the rooted connectivity augmentation problem. However, since U contains both C and D, it will appear in both instances which means that we might have to pay for covering U twice. To avoid the redundancy, we try to fix the deficient sets that will be considered by the algorithm to be those that contain exactly one core. In other words, we want to apply Frank's algorithm only for covering one halo-family. This guarantees that we will not pay for the same deficient set twice. But, then we have to run the algorithm several times because the deficient sets that contains two or more cores will not be considered by the algorithm. Similar ideas of decomposing a family of deficient sets using cores and halo-families was introduced in [24].

In Section 3.9, we will present two approximation algorithms for the (S, T)-connectivity augmentation problem which achieve the same approximation guarantee of $O(\log n)$. However, the running time of the two are different. The algorithms require the same subroutines which will be described in the next three sections.

- The subroutine for computing cores.
- The subroutine for computing the halo-set of a given core.
- The subroutine for covering the halo-family of a given core.

In contrast to the algorithm in [24], our algorithms do not require the computation of halo-sets. However, for the sake of completeness, we also provide the algorithm for computing the halo-set.

3.6 Computing cores

In this section, we describe the efficient algorithm for computing all of the cores. Required in our algorithm is an efficient maximum s, t-flow algorithm. For each pair of vertices $s \in S$ and $t \in T$,

we construct a flow network N with the source s and the sink t. We assign unit capacity to all edges in the flow network, that is, all edges have capacity 1. We then compute the maximum s,t-flow f on this network. We define the *residual digraph* with respect to a flow f to be $R_f = (V, \hat{E})$, where $\hat{E} = \{(u,v) \in E_\ell : f(u,v) < 1 \text{ or } f(v,u) > 0\}$. If the value of the maximum s,t-flow is less than $\ell + 1$, then we compute the set of vertices reachable from s in R_f, denoted by $C_{s,t}$; otherwise, we ignore this pair of vertices. By Menger's Theorem (or Max-Flow Min-Cut Theorem), we have that $C_{s,t}$ includes s but excludes t, and $d^{\text{out}}(C_{s,t}) = \ell$. In other words, $C_{s,t}$ is a deficient set. We keep the information of all deficient sets found. Note that the number of deficient set $C_{s,t}$ for all pairs of vertices $s \in S$ and $t \in T$ is at most $|S||T|$.

Let $\mathcal{C}^{pre} = \{C_{s,t} : s \in S, t \in T, d^{\text{out}}(C_{s,t}) < \ell + 1\}$ denote the family of deficient sets $C_{s,t}$ for all $s \in S$ and $t \in T$ computed by the above algorithm. In the final step, we remove from $\mathcal{C}^{pre}$ a deficient set $C_{s,t}$ that contains other deficient set $C_{s',t'} \in \mathcal{C}^{pre}$. If $C_{s,t} = C_{s',t'}$, then we keep only one copy. We denote the final family of the deficient sets by $\mathcal{C}$. We claim that $\mathcal{C}$ is a set of all the cores in the digraph.

Lemma 3.6.1. *The algorithm given above computes all of the cores in the digraph, that is, $\mathcal{C}$ is the family of all of the cores in the digraph.*

Proof. For each pair of vertices $s \in S$ and $t \in T$ such that $C_{s,t} \in \mathcal{C}^{pre}$, by the construction, since the value of the maximum s,t-flow is less than $\ell + 1$, we have that $d^{\text{out}}(C_{s,t}) < \ell + 1$. Moreover, since $s \in C_{s,t}$ and $t \in V - C_{s,t}$, this implies that $C_{s,t}$ is a deficient set for all $C_{s,t} \in \mathcal{C}^{pre}$.

We will now show that all of the cores are in $\mathcal{C}^{pre}$. We claim that $C_{s,t}$ is (inclusionwise) minimal among deficient sets that include s and exclude t. If not, then there must be a minimal such set, say C', so that $C_{s,t} - C' \neq \emptyset$. But, then removing $\delta^{\text{out}}(C')$ disconnects s and some vertices in $C_{s,t} - C'$ which implies that a vertex $v \in C_{s,t} - C'$ is not reachable from $C_{s,t}$ in the residual digraph, a contradiction. It follows that every core C is in $\mathcal{C}^{pre}$ because C separates some pair of vertices $s \in S$ and $t \in T$, and $d^{\text{out}}(C) < \ell + 1$.

In the final step, we remove from $\mathcal{C}^{pre}$ every deficient set $C_{s,t}$ contained in the other. Since any core C contains no other deficient sets, C is not removed from $\mathcal{C}^{pre}$. Since any deficient set $C_{s,t}$ must contain at least one core, if $C_{s,t}$ is not a core, it must contain a core $C_{s',t'} \in \mathcal{C}^{pre}, C_{s',t'} \neq C_{s,t}$. Hence, we conclude that, at the termination, $\mathcal{C}$ is the family of all the cores in the digraph. □

The above lemma also gives the upper bound on the number of cores.

Corollary 3.6.2. *The number of cores in the digraph is at most $|S| \cdot |T|$.*

We remark that the upper bound on the number of cores in Corollary 3.6.2 is tight. The tight example is as follows. We start by an empty digraph $G = (V, \emptyset)$, where $V = S \cup T$. To construct the initial digraph, we add an edge from each vertex $s \in S$ to each vertex $t \in T$. Clearly, the initial digraph is now 1-(S,T)-connected. To make it into the setting of the (S,T)-connectivity augmentation problem, we set the connectivity requirement to be $k = 2$. Then the cores in this digraph are of the form $\{s_i\} \cup T - \{t_j\}$ for $i = 1, 2, \ldots, |S|$ and $j = 1, 2, \ldots, |T|$. Thus, the number of cores in this digraph is $|S| \cdot |T|$, meeting the upper bound.

Running time: The running time of the algorithm depends on the running time of the maximum s,t-flow computation. Let $t(m,n)$ denote the time for computing the maximum s,t-flow. We assume that $t(m,n) = \Omega(m+n)$. Note that the time for computing a set of vertices reachable from a specified vertex is $O(m+n)$ which is dominated by the time for computing the maximum s,t-flow. Thus, the time for computing all deficient sets in $\mathcal{C}^{pre}$ is $O(|S||T| \cdot t(m,n)) = O(n^2 \cdot t(m,n))$.

The time for removing non-cores from the family $\mathcal{C}^{pre}$ depends on the implementation. With a naive implementation, the running time is $O(n^5)$. This is because we have to check whether each deficient set in $\mathcal{C}^{pre}$ is contained in the other which takes $O(n)$ times, and we have $O(|S|^2|T|^2) = O(n^4)$ number of pairs because there are $O(|S||T|)$ deficient sets in $\mathcal{C}^{pre}$. Thus, the time for computing all of the cores is $O(n^5 + n^2 \cdot t(m,n))$.

A faster implementation

With more care, we can improve the running time for removing non-cores from the family $\mathcal{C}^{pre}$. We start by proving the following proposition.

Proposition 3.6.3. *Consider a deficient set $C_{s,t} \in \mathcal{C}^{pre}$. $C_{s,t}$ is a core if and only if there is no deficient set $C_{s',t} \in \mathcal{C}^{pre}$ such that $s' \in C_{s,t}$ and $s \notin C_{s',t}$.*

Proof. First, recall that for any $s \in S$ and $t \in T$ such that the deficient set $C_{s,t}$ exists, the value of a maximum flow from s to t is ℓ. The forward direction is straightforward. Suppose $C_{s,t}$ is a core. Assume a contradiction that there is a deficient set $C_{s',t} \in \mathcal{C}^{pre}$ such that $s' \in C_{s,t}$ and $s \notin C_{s',t}$. Notice that both the bodies and the shadows of $C_{s,t}$ and $C_{s',t}$ are intersecting. Hence, by Lemma 3.5.1 (Uncrossing Lemma), $C_{s,t} \cap C_{s',t}$ is a deficient set which is properly contained in $C_{s,t}$, a contradiction.

To see the converse, suppose $C_{s,t}$ is not a core. Then $C_{s,t}$ must properly contain a core $C_{s',t'} \in \mathcal{C}^{pre}$. It is clear that $s' \in C_{s,t}$. We will show that $s \notin C_{s',t}$. The key claim is that, for any pair of vertices $\hat{s} \in S$ and $\hat{t} \in T$, if $\hat{s}$ is in the body of $C_{s',t'}$ and $\hat{t}$ is in the shadow of $C_{s',t'}$, then $C_{\hat{s},\hat{t}}$ and $C_{s',t'}$ define the same core, that is, $C_{\hat{s},\hat{t}} = C_{s',t'}$.

We will now prove the claim. Let $\hat{s}$ be any vertex in the body of $C_{s',t'}$, that is, $\hat{s} \in C_{s',t'} \cap S$. Let $\hat{t}$ be any vertex in the shadow of $C_{s',t'}$, that is, $\hat{t} \in T - C_{s',t'}$. Then after removing $\delta^{\text{out}}(C_{s',t'})$, the digraph has no $\hat{s} \to \hat{t}$ dipath. Recall that we construct $C_{\hat{s},\hat{t}}$ by taking all vertices reachable from $\hat{s}$ in the residual digraph with respect to the maximum flow from $\hat{s}$ to $\hat{t}$. This means that $C_{\hat{s},\hat{t}} \subseteq C_{s',t'}$. But, $C_{\hat{s},\hat{t}}$ cannot be properly contained in $C_{s',t'}$ because $C_{s',t'}$ is a core. Hence, $C_{\hat{s},\hat{t}} = C_{s',t'}$.

Recall to the assumption that $C_{s',t'}$ is a core properly contained in $C_{s,t}$. This means that t is in a shadow of $C_{s',t'}$; hence, $C_{s',t} = C_{s',t'}$. Similarly, since $C_{s',t'} \subsetneq C_{s,t}$, s cannot be in the body of $C_{s,t}$; otherwise, $C_{s,t} = C_{s',t'}$. Thus, $s' \in C_{s,t}$ and $s \notin C_{s',t}$ as required. □

We apply the above proposition to design an algorithm. For each deficient set $C_{s,t} \in \mathcal{C}^{pre}$, we iterate on all vertex $s' \in C_{s,t} \cap S$ to check whether there is a deficient set $C_{s',t}$ such that $s' \in C_{s,t}$ and $s \notin C_{s',t}$; if such deficient set exists, then we remove $C_{s,t}$ from $\mathcal{C}^{pre}$.

In the implementation, we use look-up table as a data structures. Precisely, we have an $|S| \times |T| \times |V|$ table, namely, L for sets in $\mathcal{C}^{pre}$, where $L(s,t,v) = 1$ if $v \in C_{s,t}$, and otherwise, $L(s,t,v) = 0$. We have an $|S| \times |T|$ table, namely, M for the family $\mathcal{C}^{pre}$, where $M(s,t) = 1$ if $C_{s,t} \in \mathcal{C}^{pre}$, and otherwise, $M(s,t) = 0$.

We will now analyze the running time of the algorithm. Initially, we have to create data structures for the family $\mathcal{C}^{pre}$ and deficient sets $C_{s,t} \in \mathcal{C}^{pre}$. Clearly, it requires $O(n^3)$ time to build the data structures. Testing whether an element is in the set requires $O(1)$. Consider the above algorithm. We have to iterate on $O(n^2)$ deficient sets in $C_{s,t} \in \mathcal{C}^{pre}$; in each iteration, we have to iterate on $O(n)$ vertices of S to test whether there exists a deficient set $C_{s',t}$ that satisfies conditions in Proposition 3.6.3. Hence, the time required for removing non-cores from $\mathcal{C}^{pre}$ is $O(n^3)$. Thus, the time for computing all of the cores is $O(n^3 + n^2 \cdot t(m,n))$.

Note that this implementation is faster than the previous one if the running time of the maximum s,t flow computation is strictly less than $O(n^3)$. The subroutine that computes all the cores is needed in other subroutines and main algorithms. However, even with the naive implementation, the time for computing all the cores is dominated by the time required by other subroutines. Thus, in the later sections, we will refer to the naive implementation of the algorithm.

3.7 Computing halo-sets

Although we cannot explicitly find the halo-family of the given core C, we can compute the union of all the deficient sets in the halo-family which is the halo-set. In this section, we describe the algorithm for computing the halo-set of a given core. This algorithm is not used by our main algorithms, but we have included it since it may be of independent interest.

We first assume that we have identified all of the cores. Let C be the given core. To compute the halo-set of C, we run the testing algorithm as follows. We start by picking an arbitrary vertex $r \in \mathrm{Body}(C)$. For each vertex $v \in V - C$, we add an auxiliary edge (r,v) to the digraph $G = (V, E_0)$, resulting in the auxiliary digraph $G + (r,v)$. We then compute the cores containing r in the auxiliary digraph $G + (r,v)$. Note that a core in the auxiliary digraph $G + (r,v)$ is also a deficient set in the original digraph G. Hence, if there is a core C' of $G + (r,v)$ containing C and containing no other cores of G, then C' must be a deficient set in $\mathrm{Halo}(C)$, and thus, v must be in the halo-set $\mathrm{H}(C)$. If there is such a deficient set C' certificating that v is in the halo-set $\mathrm{H}(C)$, then we include v in the halo-set; otherwise, we reject v.

The following lemma shows the correctness of our algorithm.

Lemma 3.7.1. *Consider a digraph G. Let C be a core in the digraph G. Let $r \in \mathrm{Body}(C)$ and $v \in V - C$. Then v is in the halo-set of C if and only if there is a core C' of the auxiliary digraph $G + (r,v)$ that contains C and v but contains no other cores of G.*

Proof. ($\Rightarrow$) Suppose v is in the halo-set of C. Then there is a deficient set C' of G such that $v \in C'$ and $C' \in \mathrm{Halo}(C)$. In other words, C' contains C and v but contains no other cores of G. Clearly,

C' remains a deficient set in $G + (r, v)$ because C' contains both r and v which implies that (r, v) does not cover C'. Moreover, every deficient set in $G + (r, v)$ that contains r must also contain v; otherwise, it would have been covered by (r, v). Assuming the minimality of C', we have that C' is a core of the auxiliary digraph $G + (r, v)$.

($\Leftarrow$) Suppose there is a core C' in the auxiliary digraph $G + (r, v)$ that contains C and v but contains no other cores of G. It is clear that C' is also a deficient set in the original digraph G. Since C' contains no cores of G other than C, C' must be a deficient set in $\mathrm{Halo}(C)$. Thus, v is in the halo-set of C. □

Running time: The running time for computing the halo-set of the given core C is straightforward. For each vertex $v \in V - C$, we have to test whether v is in the halo-set of C. Indeed, we have to find an inclusionwise minimal deficient set that contains both C and v. As discussed in Section 3.6, this can be done by computing the maximum s, t-flow from a vertex $s \in C \cup \{v\}$ to each vertex $t \in T - C \cup \{v\}$. Hence, the running time of the algorithm is $O(n^2 \cdot t(m, n))$, where $t(m, n)$ is the time for computing the maximum s, t-flow.

3.8 Covering halo-family via padded-Frank's algorithm

In this section, we describe the key subroutine in our algorithm. We use the algorithm due to Frank in Theorem 3.4.7 as a subroutine; alternatively, we can also use the more general one in Theorem 3.4.6. The proof below follows from the correctness of Frank's algorithms.

Consider a given core C, and its halo-family $\mathrm{Halo}(C)$. To cover $\mathrm{Halo}(C)$, we first add so-called "padding-edges" that covers all deficient sets not in $\mathrm{Halo}(C)$. In particular, for each core D distinct from C, we choose an arbitrary vertex $u_D \in D \cap S$ and add a padding edge from u_D to each vertex $v \in \mathrm{Shadow}(D)$, that is, the set of padding edges for D is $\{(u_D, v) : v \in \mathrm{Shadow}(D)\}$. After adding all the padding edges, we choose an arbitrary root vertex $r_C \in C \cap S$ and run Frank's algorithm on the padded digraph to solve the rooted connectivity augmentation problem with the root r_C and the terminal set T. See Figure 3.6.

The following lemma shows the correctness of the algorithm.

Lemma 3.8.1 (Padding Lemma)**.** *Let C be the chosen core. Then the set of augmenting edges F_C found by the above algorithm covers the halo-family of C, that is, every deficient set in the halo-family of C is covered by F_C.*

Proof. Let $\Pi(C)$ denote the family of deficient sets of the instance of the rooted connectivity augmentation problem in the padded digraph, that is,

$$\Pi(C) = \{U \subseteq V : r_C \in U, T - U \neq \emptyset, d^{\mathrm{out}}(U) = \ell\}$$

We claim that $\mathrm{Halo}(C) = \Pi(C)$. We first show that $\mathrm{Halo}(C) \subseteq \Pi(C)$. Consider a deficient set $U \in \mathrm{Halo}(C)$. Clearly, $r_C \in C \subseteq U$, and $T - U \neq \emptyset$. If U is not a deficient set in the instance

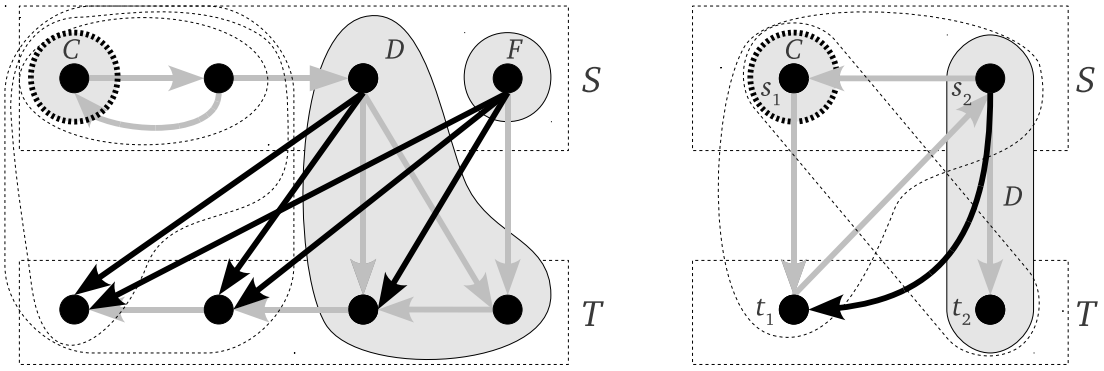

Figure 3.6: The figure shows two instances of the k-(S,T)-connectivity problem, where the initial digraph is 1-(S,T)-connected. The digraphs in the figure are padded by the padded-Frank algorithm. The grey lines denote the edges of the initial digraph. The black lines denote the padding edges. The right figure illustrates the extreme case that the core is contained in the halo-set of the other core. The cores in the right instance are $C = \{s_1\}$ and $D = \{s_2, t_2\}$. The deficient sets in the halo-family of C are $\{s_1\}$, $\{s_1, t_2\}$, and $\{s_1, s_2, t_1\}$. The halo-set of C is $\{s_1, s_2, t_1, t_2\}$ which contains all vertices of the digraph. The halo-set of C also contains D.

of the rooted connectivity augmentation problem, then there exists a padding edge $e = (v, w)$ that covers U, that is, $v \in \mathrm{Body}(U)$ and $w \in \mathrm{Shadow}(U)$. By the construction of the padding edges, there must exist a core $D \neq C$ such that $v \in \mathrm{Body}(D)$ and $w \in \mathrm{Shadow}(D)$. This implies that

$$v \in C \cap D \cap S \neq \emptyset \text{ and } w \in (T - C) \cap (T - D) \neq \emptyset.$$

In other words, both the bodies and the shadows of D and U are not disjoint, contradicting to Lemma 3.5.2.

We will now complete the proof by showing that $\Pi(C) \subseteq \mathrm{Halo}(C)$. Consider a deficient set $U \in \Pi(C)$. Then no padding edges covers U. Hence, by the construction, U contains no core distinct from C. To see this, suppose U contains a core D distinct from C. Then $\mathrm{Shadow}(U) = T - U \subseteq T - D = \mathrm{Shadow}(D)$. This implies that there exists a padding edge (u_D, v) where

$$u_D \in \mathrm{Body}(D) \subseteq \mathrm{Body}(U) \text{ and } v \in \mathrm{Shadow}(U) \subseteq \mathrm{Shadow}(D).$$

Hence, $U \in \mathrm{Halo}(C)$, proving that $\mathrm{Halo}(C) = \Pi(c)$. □

Running time: We will now analyze the time for covering the halo-family of a given core. We may assume that all of the cores have been computed. Implicitly required in this algorithm is

the efficient algorithm for computing maximum s,t-flow. Since the number of the padding edges is at most $|S||T|$, the running time for adding these edges is $O(n^2)$. The next step is to apply Frank's algorithm in Theorem 3.4.7 which runs in $O(n^2m+n\cdot t(m,n))$ time. We may assume that $m=\Omega(n)$ because all isolated vertices can be removed. Thus, the running time of the algorithm is $O(n^2+n^2m+n\cdot t(m,n))=O(n^2m+n\cdot t(m,n))$.

3.9 Approximation algorithms for the (S,T)-connectivity augmentation problem

In this section, we present the approximation algorithms for the (S,T)-connectivity augmentation problem. Intuitively, we increase (S,T)-connectivity of the digraph by iteratively adding some edges to decrease the number of cores until there are no cores left. We have two different approximation algorithms that yield the same approximation guarantee and establish the following theorem.

Theorem 3.9.1. *There exists an $O(\log n)$-approximation algorithm for the (S,T)-connectivity augmentation problem.*

3.9.1 Approximation Algorithm I: decrease the number of cores by one

The first approximation algorithm decreases the number of cores by one in each iteration. Consider any iteration of the algorithm. For each core C, we apply the subroutine (padded-Frank algorithm) in Section 3.8 to compute the set of edges $F(C)$ that covers $\mathrm{Halo}(C)$. We then choose the core C^* such that the cost of $F(C^*)$ is minimum, that is,

$$C^*=\mathrm{argmin}\{\mathrm{c}(F(C)) : C \text{ is a core.}\}$$

We add the set of edges $F(C^*)$ to the current digraph and continue to the next iteration. We repeat the process until there are no cores left in the current digraph. Observe that if there are no cores, then the (S,T)-connectivity of the digraph is at least $\ell+1$.

We will now show the correctness of the algorithm. Let $\mathcal{G}$ denote the current digraph. The key claim that shows the correctness of our algorithm is that the number of cores of $\mathcal{G}+F(C^*)$ is strictly less than the number of cores of $\mathcal{G}$. In the previous literature [24, 55, 62], the arguments relied on the fact that the cores are disjoint; however, in this case, the cores need not to be disjoint. Particularly, after adding the set of augmenting edges, we may have $j\geq 2$ new cores that intersect each other, but the union contains less than j old cores. See Figure 3.9.1.

The next lemma shows the correctness of the algorithm.

Lemma 3.9.2. *Let $\mathcal{G}$ be the current digraph. Let C be any core, and let $F(C)$ be the set of edges found by the padded-Frank algorithm. Then the number of cores in $\mathcal{G}+F(C)$ is strictly less than the number of cores in $\mathcal{G}$. In other words, the padded-Frank algorithm causes the number of cores to decrease by one.*

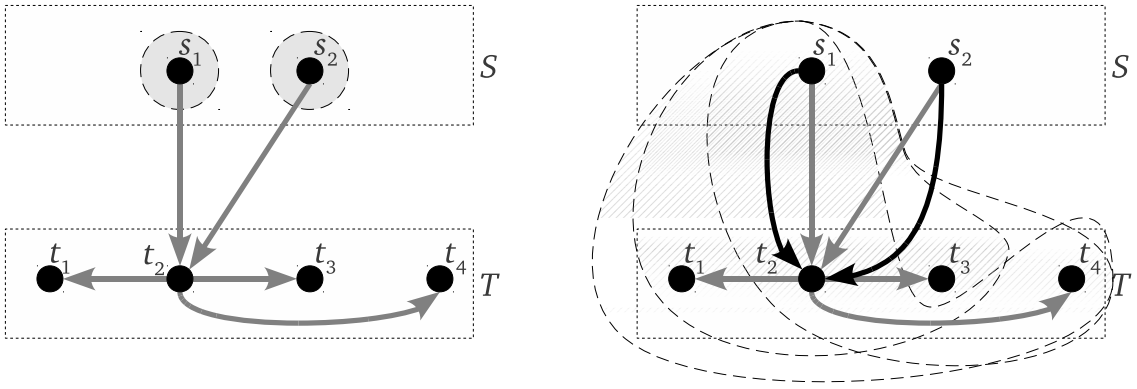

Figure 3.7: The figure shows two instances of the k-(S,T)-connectivity problem, where the initial digraph is 1-(S,T)-connected. The grey lines denote the edges of the initial digraph. The cores in the left instance are $\{s_1\}$ and $\{s_2\}$. In the right instance, $\{s_1\}$ and $\{s_2\}$ are covered by the black edges. The cores after adding the black edges are $\{s_1, t_1, t_2, t_3\}$, $\{s_1, t_1, t_2, t_4\}$, $\{s_1, t_2, t_3, t_4\}$, $\{s_2, t_1, t_2, t_3\}$, $\{s_2, t_1, t_2, t_4\}$, and $\{s_2, t_2, t_3, t_4\}$.

Proof. We will refer to the cores in the current digraph $\mathcal{G}$ as the *old cores* and refer to the cores in the new digraph $\mathcal{G} + F(C)$ as the *new cores*.

The followings are the key facts in our proof.

1. $F(C)$ covers $\mathrm{Halo}(C)$, that is, every deficient set of $\mathcal{G}$ in $\mathrm{Halo}(C)$ is covered by some edge $e \in F(C)$.

2. Every old core other than C is preserved, that is, except for C, all of the old cores are new cores.

The first fact holds because the padded-Frank algorithm find a set of edges that covers $\mathrm{Halo}(C)$. So, we only need to prove the second one. Assume a contradiction that the second fact does not hold, that is, there exists an old core $D \neq C$ that is not a new core. This means that $F(C)$ has an edge e that covers D. Note that the padded-Frank algorithm find a set of augmenting edges $F(C)$ of minimum cost that covers $\mathrm{Halo}(C)$. Hence, the augmenting edge $e \in F(C)$, which has positive cost, must cover some deficient set $U \in \mathrm{Halo}(C)$ (of the old digraph). Observe that an augmenting edge e can cover two deficient sets only if they have common vertices in both the bodies and the shadows. This is because all augmenting edges have tails in S and heads in T. It follows that U and D have intersection in both the bodies and the shadows, a contradiction to Lemma 3.5.2 (disjointness property).

Now that we have the two facts, we will prove the main statement. Observe that by the second fact, if the number of cores of $\mathcal{G} + F(C)$ is not less that the number of cores of $\mathcal{G}$, then there must

be the new core D' introduced in $\mathcal{G}+F(C)$, that is, D' is not an old core. Note that D' is a deficient set in the old digraph $\mathcal{G}$, and thus, D' must contain one or more old cores. But, D' cannot properly contain any old core $D \neq C$ because, by the fact (2), every old core $D \neq C$ is also a new core; otherwise, D' would not be inclusionwise minimal. Indeed, D' contains no core $D \neq C$. This means that D' must contain C and contain no other old core; in other words, $D' \in \mathrm{Halo}(C)$. This contradicts to the fact (1) because D' must have been covered by $F(C)$.

Thus, we conclude that there is no new core introduced in the new digraph $\mathcal{G} + F(C)$ which implies that the number of cores in $\mathcal{G}+F(C)$ is strictly less than the number of cores in $\mathcal{G}$, proving the lemma. □

Running time: We first analyze the time needed for each iteration. At the beginning of each iteration, we have to compute all the cores which takes $O(n^5 + n^2 \cdot t(m,n))$ time, where $t(m,n)$ denote the time for computing the maximum s,t-flow. For each core C, we have to compute the set of edges $F(C)$ that covers $\mathrm{Halo}(C)$ which takes $O(n^2m + n \cdot t(m,n))$ time. Since the number of cores is upper bounded by $O(n^2)$, the total time becomes $O(n^4m + n^3 \cdot t(m,n))$ which dominates the time for computing all the cores.

The number of iterations is at most the number of cores, which is $O(n^2)$, because we have to decrease the number of cores down to zero. Thus, the running time of the algorithm is $O(n^6m + n^5m \cdot t(m,n))$.

3.9.2 Cost analysis by decomposing the fractional optimal solution

In this section, we will analyze the cost incurred by the algorithm in the previous section. We start by giving the decomposition of any feasible solution to the LP of the (S,T)-connectivity augmentation problem (See Table 3.2). Indeed, our decomposition theorem holds for every feasible solution to the (S,T)-connectivity augmentation problem.

Let $\boldsymbol{x}$ denote some fixed optimal (fractional) solution to the LP for the (S,T)-connectivity augmentation problem. From this point, we will override the notation of E^* to denote the set of edges restricted to the support of $\boldsymbol{x}$, that is, $E^* = \{e \in E : \boldsymbol{x}_e > 0\}$. Also, we will override opt to denote $\mathrm{opt} = \sum_{e \in E^*} \mathrm{c}(e)\boldsymbol{x}_e$.

We say that a set of edges $F \subseteq E^*$ *fractionally covers* a deficient set U if $\boldsymbol{x}(\delta^{\mathrm{out}}_{F-E_\ell}(U)) \geq 1$. Similarly, consider any core C. We say that F *fractionally covers* the halo-family of the core C if F fractionally covers every deficient set in $\mathrm{Halo}(C)$.

Lemma 3.9.3 (Decomposition Lemma)**.** *Let $C_1, C_2, \ldots, C_t$ denote all of the cores. For each core C_i, $1 \leq i \leq t$, let $E^*(C_i)$ denote an (inclusionwise) minimal subset of E^* that fractionally covers $\mathrm{Halo}(C_i)$. Then $E^*(C_i)$ and $E^*(C_j)$ are disjoint for all $1 \leq i \neq j \leq t$. Furthermore, $\sum_{i=1}^{t} \sum_{e \in E^*(C_i)} \mathrm{c}(e)x_e \leq \mathrm{opt}$.*

Proof. We prove the first statement by a contradiction argument. Suppose to a contrary that there exists i, j with $1 \leq i \neq j \leq t$, such that $E^*(C_i) \cap E^*(C_j)$ contains an edge e. Then by the minimality of $E^*(C_i)$ and $E^*(C_j)$, e must cover some deficient set $U \in \mathrm{Halo}(C_i)$ and some deficient set $W \in \mathrm{Halo}(C_j)$. This means that e has tail in $\mathrm{Body}(C_i) \cap \mathrm{Body}(C_j)$ and head in $\mathrm{Shadow}(C_i) \cap \mathrm{Shadow}(C_j)$. In other words, both the bodies and the shadows of C_i and C_j intersect, a contradiction to Lemma 3.5.2 (disjointness property).

The second statement immediately follows from the the first one. Indeed, since $E^*(C_i)$ and $E^*(C_j)$ are disjoint for $1 \leq i \neq j \leq t$, we have

$$\sum_{i=1}^{t} \sum_{e \in E^*(C_i)} \mathrm{c}(e) x_e \leq \sum_{e \in E^*} \mathrm{c}(e) x_e = \mathrm{opt}$$

□

The next lemma compares the cost of the integral solution obtained by the padded-Frank algorithm to that of the optimal solution to the LP of the (S, T)-connectivity augmentation problem. Note that the similar argument is used in [24].

Lemma 3.9.4. *For any core C, let $F(C)$ denote the set of augmenting edges found by the padded-Frank algorithm, and let $E^*(C)$ be defined as in the previous lemma. Then $\mathrm{c}(F(C)) \leq \sum_{e \in E^*(C)} \mathrm{c}(e) x_e$.*

Proof. This follows from Theorem 3.4.7 (or Theorem 3.4.6). In fact, Frank proves that the LP relaxation for the rooted connectivity augmentation problem has an integral optimal solution. Hence, the cost of the optimal fractional solution to the LP and the cost of the optimal (integral) solution to the rooted connectivity augmentation problem are the same.

Recall that $F(C)$ is the set of augmenting edges with minimum cost that covers $\mathrm{Halo}(C)$. Thus, any feasible solution to the LP for the rooted connectivity augmentation problem (in the padded-Frank algorithm) has cost at least $\mathrm{c}(F(C))$. Since, for $i = 1, 2, \ldots, t$, $E^*(C_i)$ fractionally covers $\mathrm{Halo}(C_i)$, it implies that the $\mathrm{c}(F(C)) \leq \sum_{e \in E^*(C)} \mathrm{c}(e) x_e$. □

Lemma 3.9.5. *Let t be the number of cores, and let C^* be a core such that the edges set $F(C^*)$ has minimum cost, that is, $F(C^*) = \min_{i=1}^{t} \{\mathrm{c}(F(C)\}$. Then*

1. $\sum_{i=1}^{t} \mathrm{c}(F(C_i)) \leq \mathrm{opt}$.
2. $\mathrm{c}(F(C^*)) \leq \mathrm{opt}/t$.

Proof. For all the core $C_1, C_2, \ldots, C_t$, by Lemma 3.9.4, we have

$$\mathrm{c}(F(C_i)) \leq \sum_{e \in E^*(C_i)} \mathrm{c}(e) x_e.$$

Hence, by Lemma 3.9.3, we have

$$\sum_{i=1}^{t} \mathrm{c}(F(C_i)) \leq \sum_{i=1}^{t} \sum_{e \in E^*(C_i)} \mathrm{c}(e) x_e \leq \mathrm{opt}.$$

This proves the first statement. To proves the second one, we apply the averaging argument, that is,

$$\mathrm{c}(F(C^*)) = \min_{i=1}^{t} \{\mathrm{c}(F(C_i))\} \leq \frac{1}{t} \sum_{i=1}^{t} \mathrm{c}(F(C_i)) = \frac{\mathrm{opt}}{t}.$$

□

Corollary 3.9.6. *Total cost incurred by the algorithm is* $O(\log n)\mathrm{opt}$.

Proof. Let t_0 denotes the number of cores at the start of the algorithm. Since the number of cores in each iteration decreases by one, and the set of edges added has cost at most opt/t', where t' is the number of cores at the start of the iteration. This implies that the total cost of the edges added by the algorithm is at most

$$\left(\frac{1}{t} + \frac{1}{t-1} + \ldots + 1\right) \mathrm{opt} = O(\log t) \cdot \mathrm{opt} = O(\log n) \cdot \mathrm{opt}.$$

We note that the last equation follows from the fact that the number of cores is at most $|S||T| = O(n^2)$ proved in Corollary 3.6.2. □

3.9.3 Approximation Algorithm II: decrease the number of cores by a factor of two.

In this section, we present the second approximation algorithm. In each iteration, for each core C, we compute the minimum cost set of edges $F(C)$ that covers $\mathrm{Halo}(C)$ by the padded-Frank algorithm. The difference form the previous algorithm is that we add all of the edges found by the padded-Frank algorithm, that is, $\bigcup\{F(C) : C \text{ is a core}\}$, to the digraph. We repeat this process until there is no core left.

We claim that the algorithm terminates within $O(\log n)$ iterations. Roughly speaking, in each iteration, the algorithm adds a set of edges that causes the number of cores to decrease by a factor of two. Similar methods are discussed in [62] and implicitly in [24]. Unfortunately, since the cores in our problem need not be disjoint, the proof is more complex. We introduce a new notion that defines the progress of the algorithm. We claim that *the maximum number of body-disjoint cores* decreases by a factor of two in each iteration. Thus, the number of iteration is at most $O(\log n)$.

The following lemmas prove our claim.

Lemma 3.9.7. *No deficient set contains two cores whose bodies are intersecting.*

Proof. By Lemma 3.5.2 (disjointness property), any two distinct cores C and D whose bodies are intersecting must have disjoint shadows. Hence, $C \supseteq T - D$ and $D \supseteq T - C$. This implies that $C \cup D$ contains T, and thus, $C \cup D$ cannot be a deficient set. □

Lemma 3.9.8. *In each iteration, the maximum number of body-disjoint cores decreases by a factor of two.*

Proof. Let ν and ν' denote the maximum number of body-disjoint cores at the beginning and the end of the iteration, respectively.

We refer to cores at the beginning of the iteration as old cores and those at the end of the iteration as new cores. In each iteration, the algorithm covers all deficient sets that are contained in some halo-family. So, there is no deficient set left that contains exactly one old core. In other words, any new core contains at least two old cores. By Lemma 3.9.7, since new cores are deficient sets in the old digraph, they cannot contain two old cores whose bodies are intersecting. So, ν' body-disjoint new cores must contain at least $2\nu'$ body-disjoint old cores. Hence, $2\nu' \leq \nu$ which proves the Lemma. □

Lemma 3.9.9. *The algorithm terminates within $O(\log n)$ iterations. It runs in polynomial time. Moreover, the total cost incurred by the algorithm is at most $O(\log n)\mathrm{opt}$.*

Proof. First, the maximum number of body-disjoint cores is at most $O(|S|) = O(n)$. By Lemma 3.9.8, in each iteration, the number of body-disjoint cores decreases by a factor of two. Hence, the number of iterations is $O(\log n)$. Clearly, since the padded-Frank algorithm runs in polynomial time, the running time of the algorithm is also polynomial, proving the second statement.

To prove the last statement, we claim that the cost of edges computed in each iteration is at most opt. To see this, let $C_1, C_2, \ldots, C_t$ be all the cores. Recall that, for $i = 1, 2, \ldots, t$, $F(C_i)$ is the minimum cost set of edges that covers $\mathrm{Halo}(C_i)$, and by Lemma 3.9.5, we have that $\sum_{i=1}^{t} \mathrm{c}(F(C_i)) \leq \mathrm{opt}$. Thus, the total cost incurred in $O(\log n)$ iterations is $O(\log n)\mathrm{opt}$. □

Running Time Recall that the running time of each iteration of the first approximation algorithm is $O(n^4 m + n^3 \cdot t(m, n))$, where $t(m, n)$ is the time for computing a maximum s, t-flow. Notice that the running time of each iteration is the same in both algorithms. However, the number of iterations of the second algorithm is $O(\log n)$. Thus, the running time of the second algorithm is $O((n^4 m + n^3 \cdot t(m, n)) \log n)$.

Chapter 4

Discussion and Open Problems

In this thesis, we studied the (S,T)-connectivity problem, and its extension, the k-(S,T)-connectivity problem. Three variants of the (S,T)-connectivity have been discussed. The first variant is the standard (S,T)-connectivity problem. We presented a 2-approximation algorithm. This approximation guarantee is the same as the current best approximation guarantee for its special case, the minimum cost strongly connected subgraph problem. Getting an approximation guarantee better than 2 implies an improved approximation guarantee for the minimum cost strongly connected subgraph problem. In [77], Vetta presents a $3/2$-approximation algorithm for the minimum size strongly connected subgraph problem. An interesting question is whether one can get an improvement on the 2-approximation algorithm for the standard (S,T)-connectivity problem where augmenting edges have unit-costs. On the negative side, Gabow, Goemans, Tardos and Williamson [41] showed that the minimum cost strongly connected subgraph problem is APX-hard.

The second variant is the relaxed (S,T)-connectivity problem. We presented an approximation algorithm whose approximation guarantee is asymptotically tight. Our approximation guarantee is the same as the approximation guarantee for the directed Steiner tree problem, which is a special case. The best known approximation algorithm for the directed Steiner tree problem acheives an approximatio guarantee of $O(\log^3 n)$ (See note in Section 2.5.1.), but the running time is quasi polynomial in the number of vertices. An open question in the area is to get an polylogarithmic-approximation algorithm for the directed Steiner tree problem that has a running time polynomial in the number of terminals.

The most general variant, the unrestricted (S,T)-connectivity problem, is as hard as the directed Steiner forest problem. In other words, getting an improvement in either the lower bound or the upper bound of the unrestricted (S,T)-connectivity problem would yield an improvement for the directed Steiner forest problem as well. The hardness of the directed Steiner forest problem comes from the maximum label cover problem. For the maximum label cover problem, Charikar, Hajiaghayi and Karloff [13] gives an $O(n^{1/3})$-approximation algorithm. Hence, it may be possible to get an $O(n^{1/3})$-approximation algorithms for both the directed Steiner forest problem and the unrestricted (S,T)-connectivity problem. If so, this will give an improvement on the current

best approximation guarantee for the directed Steiner forest problem which is $O(n^{4/5+\epsilon})$ for $\epsilon > 0$ in [27].

The k-(S,T)-connectivity problem captures some properties of the minimum cost k-vertex connected spanning subgraph problem (k-VCSS). The approximation guarantee of k-VCSS is quite open in both the upper bound and the lower bound. The k-VCSS problem is known to be APX-hard. The k-(S,T)-connectivity problem may be harder to approximate than the k-VCSS problem. On the positive side, when $k \geq 2$ and $k = O(1)$, we have an $O(\log n)$-approximation algorithm for the k-(S,T)-connectivity problem, but there exist $O(1)$-approximation algorithms for k-VCSS . Improving on our $O(\log n)$-approximation guarantee for $k = O(1)$ is an open question. A hardness of approximation result that "separates" between k-VCSS and the k-(S,T)-connectivity problem would be interesting.

References

[1] *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing, 4-6 May 1992, Victoria, British Columbia, Canada*. ACM, 1992. 59

[2] *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*. ACM, 2003. 59

[3] *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*. SIAM, 2005. 58, 61

[4] *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*. IEEE Computer Society, 2008. 57

[5] *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*. IEEE Computer Society, 2009. 57, 60

[6] László Babai, editor. *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. ACM, 2004. 60

[7] Jrgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2008. 15, 17

[8] András A. Benczúr. Pushdown-reduce: an algorithm for connectivity augmentation and poset covering problems. *Discrete Applied Mathematics*, 129(2-3):233–262, 2003. 1

[9] Gruia Calinescu and Alexander Zelikovsky. The polymatroid steiner problems. In Fleischer and Trippen [32], pages 234–245. 18

[10] Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In Ladner and Dwork [57], pages 167–176.

[11] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. In *SODA*, pages 192–200, 1998. 18

[12] Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999.

[13] Moses Charikar, MohammadTaghi Hajiaghayi, and Howard J. Karloff. Improved approximation algorithms for label cover problems. In Fiat and Sanders [31], pages 23–34. 54

[14] Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. In Teng [73], pages 532–541. 17

[15] Joseph Cheriyan, Santosh Vempala, and Adrian Vetta. Approximation algorithms for minimum-cost k-vertex connected subgraphs. In *STOC*, pages 306–312, 2002. 1

[16] Joseph Cheriyan, Santosh Vempala, and Adrian Vetta. An approximation algorithm for the minimum-cost k-vertex connected subgraph. *SIAM J. Comput.*, 32(4):1050–1055, 2003. 1, 35

[17] Julia Chuzhoy and Sanjeev Khanna. Algorithms for single-source vertex connectivity. In *FOCS* [4], pages 105–114.

[18] Julia Chuzhoy and Sanjeev Khanna. An O($k^3 \log n$)-approximation algorithm for vertex-connectivity survivable network design. *CoRR*, abs/0812.4442, 2008.

[19] Julia Chuzhoy and Sanjeev Khanna. An O($k^3 \log n$)-approximation algorithm for vertex-connectivity survivable network design. In *FOCS* [5], pages 437–441.

[20] Artur Czumaj and Andrzej Lingas. On approximability of the minimum-cost *k*-connected spanning subgraph problem. In *SODA*, pages 281–290, 1999. 33, 34

[21] Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *STOC*, pages 750–759, 1999. 6

[22] Jack Edmonds. Optimum branchings. *Journal of Research of National Bureau of Standards*, 71B(4):233–240, 1967. 2, 17

[23] Steffen Enni. A 1-(S, T)-edge-connectivity augmentation algorithm. *Math. Program.*, 84(3, Ser. B):529–535, 1999. Connectivity augmentation of networks: structures and algorithms (Budapest, 1994). 1

[24] Jittat Fakcharoenphol and Bundit Laekhanukit. An O($\log^2 k$)-approximation algorithm for the k-vertex connected spanning subgraph problem. In Ladner and Dwork [57], pages 153–158. 1, 3, 31, 35, 42, 48, 51, 52

[25] Uriel Feige. A threshold of ln *n* for approximating set cover (preliminary version). In *STOC*, pages 314–318, 1996.

[26] Uriel Feige. A threshold of ln *n* for approximating set cover. *J. ACM*, 45(4):634–652, 1998. 25

[27] Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximating algorithms for directed steiner forest. In Mathieu [61], pages 922–931. 55

[28] Sharon Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithms for directed steiner forest. *Electronic Colloquium on Computational Complexity (ECCC)*, 14(120), 2007.

[29] Cristina G. Fernandes. A better approximation ratio for the minimum k-edge-connected spanning subgraph problem. In *SODA*, pages 629–638, 1997.

[30] Cristina G. Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *J. Algorithms*, 28(1):105–124, 1998.

[31] Amos Fiat and Peter Sanders, editors. *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*. Springer, 2009. 57

[32] Rudolf Fleischer and Gerhard Trippen, editors. *Algorithms and Computation, 15th International Symposium, ISAAC 2004, HongKong, China, December 20-22, 2004, Proceedings*, volume 3341 of *Lecture Notes in Computer Science*. Springer, 2004. 56

[33] A. Frank and É. Tardos. An application of submodular flows. *Linear Algebra and its Applications*, 114/115:329–348, 1989. 15

[34] András Frank. Kernel systems of directed graphs. *Acta Sci. Math. (Szeged)*, 41:63–76, 1979. 2, 15, 17

[35] András Frank. Increasing the rooted-connectivity of a digraph by one. *Math. Program.*, 84(3, Ser. B):565–576, 1999. Connectivity augmentation of networks: structures and algorithms (Budapest, 1994). 2, 15, 17, 37

[36] András Frank. Rooted k-connections in digraphs. *Technical report published by the Egrevry Research Group*, (TR-2006-07), 2006.

[37] András Frank. Rooted k-connections in digraphs. *Discrete Applied Mathematics*, 157(6):1242–1254, 2009. 2, 15, 17, 37

[38] András Frank and Tibor Jordán. Minimal edge-coverings of pairs of sets. *J. Comb. Theory, Ser. B*, 65(1):73–110, 1995. 1

[39] Delbert Ray Fulkerson. Packing rooted directed cuts in a weighted directed graph. *Mathematical Programming*, 6(1):1–13, 1974. 2, 17

[40] Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the smallest k-edge connected spanning subgraph by lp-rounding. In *SODA* [3], pages 562–571. 1

[41] Harold N. Gabow, Michel X. Goemans, Éva Tardos, and David P. Williamson. Approximating the smallest k-edge connected spanning subgraph by lp-rounding. *Networks*, 53(4):345–357, 2009. 1, 6, 33, 34, 54

[42] Michel X. Goemans, Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, Éva Tardos, and David P. Williamson. Improved approximation algorithms for network design problems. In *SODA*, pages 223–232, 1994. 35

[43] Magnús M. Halldórsson, editor. *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*. Springer, 2000. 60

[44] Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Aravind Srinivasan, and Nan Wang. Integrality ratio for group steiner trees and directed steiner trees. In *SODA*, pages 275–284, 2003.

[45] Eran Halperin, Guy Kortsarz, Robert Krauthgamer, Aravind Srinivasan, and Nan Wang. Integrality ratio for group steiner trees and directed steiner trees. *SIAM J. Comput.*, 36(5):1494–1511, 2007.

[46] Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *STOC* [2], pages 585–594. 6

[47] Christopher S. Helvig, Gabriel Robins, and Alexander Zelikovsky. An improved approximation scheme for the group steiner problem. *Networks*, 37(1):8–20, 2001. 18

[48] Klaus Jansen, Stefano Leonardi, and Vijay V. Vazirani, editors. *Approximation Algorithms for Combinatorial Optimization, 5th International Workshop, APPROX 2002, Rome, Italy, September 17-21, 2002, Proceedings*, volume 2462 of *Lecture Notes in Computer Science*. Springer, 2002. 59

[49] Samir Khuller, Balaji Raghavachari, and Neal E. Young. Approximating the minimum equivalent digraph. *SIAM J. Comput.*, 24(4):859–872, 1995. 6

[50] Samir Khuller, Balaji Raghavachari, and Neal E. Young. Approximating the minimum equivalent digraph. *CoRR*, cs.DS/0205040, 2002.

[51] Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. In *STOC* [1], pages 759–770. 1

[52] Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994. 1

[53] Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network-design problems. In Jansen et al. [48], pages 185–199.

[54] Guy Kortsarz, Robert Krauthgamer, and James R. Lee. Hardness of approximation for vertex-connectivity network design problems. *SIAM J. Comput.*, 33(3):704–720, 2004.

[55] Guy Kortsarz and Zeev Nutov. Approximation algorithm for k-node connected subgraphs via critical graphs. In Babai [6], pages 138–145. 40, 48

[56] Guy Kortsarz and Zeev Nutov. Approximating k-node connected subgraphs via critical graphs. *SIAM J. Comput.*, 35(1):247–257, 2005. 1, 35

[57] Richard E. Ladner and Cynthia Dwork, editors. *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*. ACM, 2008. 56, 57

[58] Yuval Lando and Zeev Nutov. Inapproximability of survivable networks. *Theor. Comput. Sci.*, 410(21-23):2122–2125, 2009. 33, 34

[59] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. In *STOC*, pages 286–293, 1993.

[60] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[61] Claire Mathieu, editor. *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*. SIAM, 2009. 58, 60

[62] Zeev Nutov. An almost $o(\log k)$-approximation for k-connected subgraphs. In Mathieu [61], pages 912–921. iii, 2, 35, 48, 52

[63] Zeev Nutov. Approximating minimum cost connectivity problems via uncrossable bifamilies and spider-cover decompositions. In *FOCS* [5], pages 417–426. 35

[64] David Peleg. Approximation algorithms for the label-cover$_{\max}$ and red-blue set cover problems. In Halldórsson [43], pages 220–230.

[65] David Peleg. Approximation algorithms for the label-cover$_{\max}$ and red-blue set cover problems. *J. Discrete Algorithms*, 5(1):55–64, 2007.

[66] R. Ravi and David P. Williamson. An approximation algorithm for minimum-cost vertex-connectivity problems. In *SODA*, pages 332–341, 1995. 1, 40

[67] R. Ravi and David P. Williamson. An approximation algorithm for minimum-cost vertex-connectivity problems. *Algorithmica*, 18(1):21–43, 1997. 1, 35

[68] R. Ravi and David P. Williamson. Erratum: an approximation algorithm for minimum-cost vertex-connectivity problems. In *SODA*, pages 1000–1001, 2002. 1

[69] R. Ravi and David P. Williamson. Erratum: An approximation algorithm for minimum-cost vertex-connectivity problems. *Algorithmica*, 34(1):98–107, 2002. 1

[70] Alexander Schrijver. Min-max relations for directed graphs. In *Bonn Workshop on Combinatorial Optimization (Bonn, 1980)*, volume 16 of *Annals of Discrete Mathematics*, pages 261–280. North-Holland, Amsterdam, 1982. 1

[71] Alexander Schrijver. Supermodular colourings. In *Matroid theory (Szeged, 1982)*, volume 40 of *Colloquia Mathematica Societatis Jnos Bolyai 40*, pages 327–343. North-Holland, Amsterdam, 1985. 1

[72] Robert Endre Tarjan. A good algorithm for edge-disjoint branching. *Inf. Process. Lett.*, 3(2):51–53, 1974. 2, 17

[73] Shang-Hua Teng, editor. *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*. SIAM, 2008. 57

[74] V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[75] László A. Végh and András A. Benczúr. Primal-dual approach for directed vertex connectivity augmentation and generalizations. In *SODA* [3], pages 186–194. 1

[76] László A. Végh and András A. Benczúr. Primal-dual approach for directed vertex connectivity augmentation and generalizations. *ACM Transactions on Algorithms*, 4(2), 2008. 1

[77] Adrian Vetta. Approximating the minimum strongly connected subgraph via a matching lower bound. In *SODA*, pages 417–426, 2001. 54

[78] Alexander Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18(1):99–110, 1997. 18