Approximation algorithms for minimum-cost k-(S,T) connected digraphs

J. Cheriyan * B. Laekhanukit †

November 30, 2010

Abstract

We introduce a model for NP-hard problems pertaining to the connectivity of graphs, and design approximation algorithms for some of the key problems in this model.

One of the well-known NP-hard problems is to find a minimum-cost strongly connected spanning subgraph of a directed network. In the *minimum-cost* k-(S,T) *connected digraph* (abbreviated, k-(S,T) connectivity) problem we are given a positive integer k, a directed graph $G = (V,E)$ with non-negative costs on the edges, and two subsets S,T of V; the goal is to find a subset of edges $\widehat{E}$ of minimum cost such that the subgraph $(V,\widehat{E})$ has k edge-disjoint directed paths from each vertex in S to each vertex in T. When $k=1$ and $S=T=V$, we get the minimum-cost strongly connected spanning subgraph problem.

Our model of k-(S,T) connectivity, in its full generality, is at least as hard for approximation as the label-cover problem, even when the connectivity parameter k is one. We give a simple reduction from the directed Steiner network problem, and it is well known that the latter problem is at least as hard as the label-cover problem.

Rather than focusing on this general version of the problem, most of our results focus on a specialized version that we call the *standard* version, where every edge of positive cost has its tail in S and its head in T. This version of the problem captures NP-hard problems such as the minimum-cost k-vertex connected spanning subgraph problem. We call it the standard version because a still further specialization of it was introduced and studied by Frank and Jordan more than fifteen years ago; in their model, which is polynomial-time solvable, every edge from S to T is present and has unit cost.

One of our key contributions is an approximation algorithm with a guarantee of $O((\log k)(\log n))$ for the standard version of the k-(S,T) connectivity problem. For $k=1$, we give a simple 2-approximation algorithm that generalizes a well-known 2-approximation algorithm for the minimum-cost strongly connected spanning subgraph problem. For $k=2$, we give a simple 4-approximation algorithm, though the analysis is nontrivial.

Besides the standard version, we study another version that is intermediate between the standard version and the problem in its full generality (which is label-cover hard). In the relaxed version of the (S,T) connectivity problem, each edge of positive cost has its head in T but there is no restriction on the tail. We study the relaxed version with the connectivity parameter k fixed at one, and observe that this version is at least as hard to approximate as the directed Steiner tree problem. We match this by giving an algorithm that achieves an approximation guarantee of $\alpha(n)+1$ for the relaxed (S,T) connectivity problem, where $\alpha(n)$ denotes the best approximation guarantee available for the directed Steiner tree problem. The algorithm is simple, but not the analysis. The key to the analysis is a structural result that

*Dept. of Comb. & Opt., U. Waterloo, Waterloo ON Canada N2L 3G1. Email: jcheriyan@uwaterloo.ca

†Dept. of Computer Science, McGill University, Montreal QC Canada Email: blaekh@cs.mcgill.ca

decomposes any feasible solution into a set of so-called junction trees that are disjoint on the vertices of T. Our algorithm and analysis specialize to the case when the digraph is acyclic on T, meaning that there exists no dicycle that contains two distinct vertices of T. In this setting, we show that the relaxed (S, T) connectivity problem is at least as hard to approximate as the set covering problem, and we prove that our algorithm achieves a matching approximation guarantee of $O(\log |S|)$.

Our $O((\log k)(\log n))$ approximation guarantee for the k-(S, T) connectivity problem is obtained by applying the halo-set method, which is a versatile algorithmic paradigm for handling network design problems with vertex connectivity requirements.

1 Introduction

Classical Combinatorial Optimization focuses on fundamental models and algorithms for solving these models to optimality in polynomial time. Some examples are: matroids and the greedy algorithm for finding a minimum-cost basis; matchings (in nonbipartite graphs) and Edmonds' blossom algorithm for finding a maximum matching. This coupling of models and algorithms offers several benefits, such as versatile models that capture many special cases that arise in practice or theory, and a deeper understanding of algorithmic tools and their power, thereby leading to wider applications beyond the original models, etc.

1.1 The model of k-(S,T) connectivity

We introduce a model for NP-hard problems pertaining to the connectivity of graphs. One of the well-known NP-hard problems is to find a minimum-cost strongly connected spanning subgraph of a directed network. In the *minimum-cost* k-(S,T) *connected digraph* (abbreviated, k-(S,T) connectivity) problem we are given an integer $k \geq 0$, a directed graph $G = (V, E_0 \cup E)$ with positive costs on the edges in E, and two subsets S, T of V; we may assume that each edge in E_0 has zero cost. The goal is to find a subset of edges $\widehat{E} \subseteq E$ of minimum cost such that the subgraph $(V, E_0 \cup \widehat{E})$ has k edge-disjoint directed paths (abbreviated, dipaths) from each vertex in S to each vertex in T. When $k = 1$ and $S = T = V$, we get the minimum-cost strongly connected spanning subgraph (abbreviated, SCSS) problem.

We call E the set of *augmenting edges*, and we call $G_0 = (V, E_0)$ the *initial digraph*. The vertices in $V - (S \cup T)$ are called *optional*. The initial digraph $G_0 = (V, E_0)$ can be arbitrary. Throughout, we use n and m to denote the number of vertices and the number of edges, respectively. We use opt to denote the cost of an optimal solution, and we use E^* to denote the set of edges in an (fixed, arbitrary) optimal solution. When $k = 1$ we drop the connectivity parameter k and refer to our problem as (S,T) connectivity.

Our model of k-(S,T) connectivity, in its full generality, is at least as hard for approximation as the label-cover problem, even when the connectivity parameter k is one. There is a simple reduction from the directed Steiner network (a.k.a. directed Steiner forest) problem, see Theorem 1, and it is well known that the latter problem is at least as hard as the label-cover problem, see [1], [33, Corollary 16.39].

Rather than focusing on this general version of the problem, most of our results focus on a specialized version that we call the standard version, where every edge of positive cost has its tail in S and its head in T. This version of the problem captures NP-hard problems such as the minimum-cost k-edge connected spanning subgraph (abbreviated, k-ECSS) problem and the minimum-cost k-vertex connected spanning subgraph (abbreviated, k-VCSS) problem, which have been extensively studied in the area of approximation algorithms for almost two decades yet there are significant problems left open. We call it the standard version because a still further specialization of it was introduced and studied by Frank and Jordan more than fifteen years ago [15]. Part of their motivation was to extend their famous min-max theorem giving an optimal characterization for the vertex-connectivity augmentation problem on directed graphs to the more general setting of the k-(S,T) connectivity augmentation problem, where every edge from S to T is present and has unit cost. Thus the model introduced and studied in [15] is polynomial-time solvable; subsequently, improved algorithms were presented by [32]. Moreover, [15] proves min-max results for some of these problems; also see [29]. To the best of our knowledge, the minimum-cost version of the k-(S,T) connectivity augmentation model of [15] has not been studied before.

One of our key results is an approximation algorithm with a guarantee of $O((\log k)(\log n))$ for the standard version of the k-(S,T) connectivity problem. This algorithm is based on the algorithmic paradigm of the *halo-set method*, which was used previously in [9], building on previous work by [24], and others.

This algorithmic paradigm is surprisingly effective for problems in network design with vertex-connectivity requirements. We give an overview of this paradigm in Section 1.3, and mention some of the key results needed to extend this paradigm to the setting of standard k-(S,T) connectivity.

An immediate question is whether our approximation guarantee for k-(S,T) connectivity is optimal, or (more realistically) whether the approximation guarantee can be improved substantially. For $k = O(1)$, note that our approximation guarantee is $O(\log n)$. At the moment, it is not clear whether there exists a logarithmic (in n) hardness threshold for $k = O(1)$. For $k = 1$, we give a simple 2-approximation algorithm that generalizes a well-known 2-approximation algorithm of [17, 21] for the SCSS problem. But already for $k = 2$, there are substantial difficulties. We give a 4-approximation algorithm in Section 4; the algorithm is simple, but the analysis is nontrivial. We could not find any simple way to achieve an approximation guarantee of $O(1)$ for the 2-(S,T) connectivity problem.

Besides the standard version, we study another version that is intermediate between the standard version and the problem in its full generality (which is label-cover hard). In the relaxed version of the (S,T) connectivity problem, each edge of positive cost has its head in T but there is no restriction on the tail. We study the relaxed version with the connectivity parameter k fixed at one, and observe that this version is at least as hard to approximate as the directed Steiner tree problem. The latter problem has a hardness threshold of $\Omega(\log^{2-\epsilon} n)$ assuming that NP is not contained in ZPTIME($n^{\text{polylog } n}$), [19]. Let $\alpha(n)$ denote the best approximation guarantee available for the directed Steiner tree problem; the results of [3, 20] show that an approximation guarantee of $O(\log^3 n)$ can be achieved in time $(n^{O(\log n)})$. We give an algorithm that achieves an approximation guarantee of $\alpha(n) + 1$ for the relaxed (S,T) connectivity problem. The algorithm is simple, but not the analysis. The key to the analysis is a structural result that decomposes any feasible solution into a set of so-called junction trees that are disjoint on the vertices of T; in fact, each vertex of T appears in exactly one of these junction trees. Our algorithm and analysis specialize to the case when the digraph is acyclic on T, meaning that there exists no dicycle that contains two distinct vertices of T. In this setting, we show that the relaxed (S,T) connectivity problem is at least as hard to approximate as the set covering problem, and we prove that our algorithm achieves a matching approximation guarantee of $O(\log n)$.

Figure 1 summarizes the model via the three versions of k-(S,T) connectivity by illustrating some of the key NP-hard problems in network design captured by it.

In brief, the model of k-(S,T) connectivity captures several of the keystone problems in the area of approximation algorithms for network design, such as the SCSS problem and its extensions to k-edge connectivity, the k-VCSS problem (Section 5.1), the directed Steiner-tree problem (Theorem 1), the set covering problem (Theorem 6), and the directed Steiner network problem (Theorem 1). Despite this versatility, the model is amenable to simple algorithmic schemes that give state-of-the-art approximation guarantees; meaning that the approximation guarantees are almost as good as those for the well-studied studied special cases.

Many of the results of this paper were first presented in the second author's thesis [25]. We mention that many other types/models of connectivity problems have been studied from the perspective of approximation algorithms [5, 6, 11, 23, 27], etc., but we restrict most of our discussion to the literature that connects directly to our model.

1.2 Frank's algorithm for rooted connectivity

Frank (see [2, 16, 12, 13, 14]) gave polynomial-time algorithms and min-max theorems for finding a min-cost "rooted out-subgraph". More precisely, Frank's results focus on the special case of the min-cost

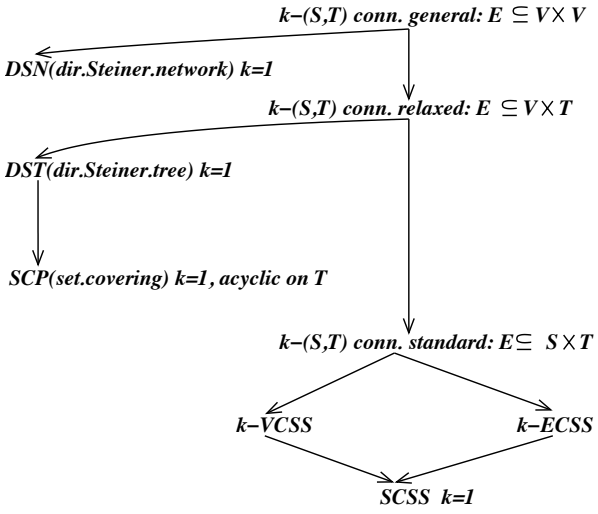


Figure 1: An illustration of the model of k-(S,T) connectivity, showing some of the key NP-hard problems in network design captured by it.

(S,T) connectivity problem where S consists of a single vertex, called the root, and every augmenting edge has its head in T, that is, $(v,w) \in E \Longrightarrow w \in T$. Note that the restriction on E is critical; without this restriction, the directed Steiner tree problem would be a special case of this problem; the directed Steiner tree problem is known to be NP-hard.

It is easily seen that Frank's result applies also for the min-cost "rooted in-subgraph" problem, by replacing each edge (v,w) by its reverse edge (w,v) and making appropriate changes for T and E. For completeness, we have stated both versions of Frank's results in Section 2, see Theorems 11 and 12.

We stress that Frank's results (see Theorem 12) immediately give an approximation guarantee of $\min\{|S|,|T|\}$ for the k-(S,T) connectivity problem. The main point of the results in our paper is to obtain substantial improvements on this approximation guarantee. (We have no results of our own on the problems addressed by Frank.)

1.3 The halo-set method

This section has an overview of the generic halo-set method, which is an effective algorithmic paradigm for problems in network design with vertex-connectivity requirements. Our presentation closely follows the method in [9], which in turn builds on previous work by [24], and others. By the (S,T) *connectivity* of a digraph we mean the maximum integer ℓ such that the digraph is ℓ-(S,T) connected, that is, the maximum ℓ such that the digraph has ℓ edge-disjoint s,t dipaths for every $s \in S$ and every $t \in T$ (we also use this term as an abbreviation for the min-cost 1-(S,T) connected digraph problem, but the context will resolve any ambiguity).

Consider the algorithmic problem of computing a subgraph of approximately minimum cost that satisfies the requirements of k-(S,T) connectivity. We start with a current graph that has no edges (in this overview, we use "graph" rather than "digraph" for readability). Then we apply k outer iterations. Each outer iteration increases the (S,T) connectivity (of the current graph) by one by adding a set of edges of approximately minimum cost. The analysis of the outer iterations applies LP-scaling and incurs a factor of $O(\log k)$ in the approximation guarantee for the k outer iterations, see [18]. In more detail, LP-scaling can be applied to

show that the cost incurred by the i-th outer iteration is $\leq \mathrm{opt}/(k+1-i)$; summing this over the k outer iterations, we get the above factor of $O(\log k)$. See [18, 28, 24, 9] for other applications of this method.

We are left with the key problem of increasing the (S,T) connectivity (of the current graph) by one, by adding a set of edges of approximately minimum cost. The halo-set method solves this by applying a number of so-called inner iterations. For the sake of readability, let us illustrate the method in the simpler setting of undirected graphs, where the current graph is a tree and the goal is to increase the edge connectivity from $\ell = 1$ to $\ell + 1 = 2$. We need a few definitions to describe an inner iteration. A *deficient set* is a set of vertices W that certifies that the current graph is not $(\ell+1)$-connected; for our example, a deficient set is a set $W \subseteq V$ such that the cut $(W, V-W)$ has size one. A *core* C means an inclusion-wise minimal deficient set; thus, C is a deficient set, but none of its proper subsets is a deficient set; for our example, every leaf vertex of the tree (current graph) forms a core. The *halo-family* of a core C, denoted $\mathrm{Halo}(C)$, is the family of all deficient sets W that contain C and contain no other core, and the *halo-set* of a core C, denote $\mathrm{H}(C)$, is the union of all the deficient sets that are members of $\mathrm{Halo}(C)$, i.e., $\mathrm{H}(C) = \bigcup\{W : W \in \mathrm{Halo}(C)\}$. For our example, let v be a leaf of the tree (current graph); then $C = \{v\}$ is a core, and $\mathrm{H}(C)$ consists of the vertex set of a maximal path starting at v, call it $v, v_2, v_3, \ldots, v_q$, such that all other vertices v_i in the path have degree two in the tree – then we have $\mathrm{H}(C) = \{v, v_2, v_3, \ldots, v_q\}$ and $\mathrm{Halo}(C) = \{C, \{v, v_2\}, \{v, v_2, v_3\}, \ldots, \{v, v_2, v_3, \ldots, v_q\}\}$.

An edge $\{u, v\}$ is said to *cover* a deficient set W if the edge has one end-vertex in W and the other end-vertex in the so-called vertex-complement W^* of W; in the case of edge-connectivity requirements, $W^* = V - W, \forall W \subseteq V$. The goal is to cover all of the deficient sets by adding a set of edges of approximately minimum cost; note that the connectivity increases by one when all of the deficient sets get covered. The inner iteration achieves this goal by repeatedly covering halo-families. A set of edges F is said to cover a halo-family $\mathrm{Halo}(C)$ if every deficient set $W \in \mathrm{Halo}(C)$ is covered by some edge in F.

Each inner iteration computes the cores and halo-sets at the start of the iteration, and covers all of the associated halo-families; the details of an inner iteration are discussed below. We repeatedly apply inner iterations, until all of the deficient sets are covered, that is, until the current graph has no cores. In some applications of the halo-set method, the number of inner iterations can be proved to be logarithmic in the maximum number of halo-families (which is the same as the maximum number of cores); let $n_{\max}(\mathcal{C})$ denote this maximum. In general, unlike our example in the previous paragraph, the cores (and halo-sets) need not be pairwise disjoint; nevertheless, if $n_{\max}(\mathcal{C})$ can be upperbounded by $n^{O(1)}$, then the number of inner iterations is $O(\log n)$, assuming the previous sentence applies.

The design and analysis of an inner iteration is specific to the problem being solved. In the case of k-(S,T) connectivity our key tool is a padded version of Frank's algorithm for min-cost k-(r,T) connectivity. It turns out that we can compute an optimal (minimum cost) cover for one halo-family, by applying this version of Frank's algorithm. One of our algorithms for k-(S,T) connectivity separately computes an optimal cover F_i for each halo-family $\mathrm{Halo}(C_i)$ and then adds the union of all these covers, $\bigcup F_i$, to the current graph. The cost analysis of this algorithm is based on our result that any optimal solution E^* can be partitioned into disjoint sets $E^*_1, E^*_2, \ldots, E^*_q$ such that E^*_i is a cover for $\mathrm{Halo}(C_i)$. Consequently, the cost incurred by an inner iteration (of this particular algorithm) is $\leq \mathrm{opt}$. Assuming that the number of inner iterations is $O(\log n_{\max}(\mathcal{C})) = O(\log n)$, we get an approximation guarantee of $O(\log n)$ for increasing the (S,T) connectivity by one.

1.4 Summary of results on k-(S,T) connectivity

This subsection summarizes our results in the model of k-(S,T) connectivity, see Sections 3–7. These results are proved under the assumption that the sets S and T are disjoint; this is without loss of generality, see Proposition 10.

Theorem 1. *Consider the k-(S,T) connectivity problem with the connectivity parameter k equal to one. The hardness of approximation of the problem depends on the version of the problem (and thus on the restrictions on the augmenting edges).*

- *The standard (S,T) connectivity problem is at least as hard as the SCSS problem.*
- *The relaxed (S,T) connectivity problem is at least as hard as the directed Steiner tree problem.*
- *The (S,T) connectivity problem (without any restrictions on the augmenting edges) is at least as hard as the directed Steiner network problem.*

We have (almost) matching approximation guarantees for the first two versions.

The next result gives a 2-approximation algorithm for the standard (S,T) connectivity problem, and the details are given in Section 3. Note that no approximation guarantee better than 2 is known for the SCSS problem which is a rather special case of our problem (standard (S,T) connectivity), although the former problem (SCSS) has been studied for almost two decades.

Theorem 2. *Consider the standard version of the (S,T) connectivity problem; thus $k = 1$. There is a 2-approximation algorithm that runs in polynomial time.*

The next result addresses the 2-(S,T) connectivity problem; the details are in Section 4.

Theorem 3. *Consider the standard version of the 2-(S,T) connectivity problem. There is a 4-approximation algorithm that runs in polynomial time.*

The next result addresses the standard version of the k-(S,T) connectivity problem; the details are in Sections 5.

Theorem 4. *There is a polynomial-time approximation algorithm for the standard version of the min-cost k-(S,T) connected digraph problem that achieves a guarantee of $O(\log k \cdot \log n)$.*

The next result gives an almost tight approximation guarantee for the relaxed version of the (S,T) connectivity problem, where each augmenting edge has its head in T but the tail is unrestricted. The guarantee is tight up to an additive term of one. This result is proved in Section 6, and so is the last result.

Theorem 5. *Consider the relaxed k-(S,T) connectivity problem with the connectivity parameter k equal to one. There exists a $(\alpha(n)+1)$-approximation algorithm, where $\alpha(n)$ denotes the (best available) approximation guarantee for the directed Steiner tree problem. In particular, there is an $O(\log^3 n)$-approximation algorithm that runs in quasi-polynomial time.*

We have some results on the relaxed (S,T) connectivity problem on acyclic digraphs. See Section 2 for precise definitions.

Theorem 6. *Consider the special case of the relaxed (S,T) connectivity problem (thus $k = 1$) where the given digraph G is acyclic on T. This problem is at least as hard as the set covering problem. Moreover, there exists an $O(\log |S|)$-approximation algorithm for this problem.*

1.5 Organization of the paper

The above discussion gives a quick tour of all of the results in our paper. Sections 3 and 4 give our approximation algorithms and analysis for the standard versions of (S,T) connectivity and 2-(S,T) connectivity, respectively. Sections 5 gives our poly-logarithmic approximation algorithm for standard k-(S,T) connectivity. Sections 6 gives our approximation algorithm for the model of relaxed (S,T) connectivity. Some of the definitions and high-level explanations given in Section 1 may be repeated below.

2 Preliminaries

Most of our notation and terms are standard, see e.g., Bang-Jensen & Gutin [2], Diestel [7], or Schrijver [30]. When we say that two sets S_1, S_2 intersect (or, are intersecting), then we mean that $S_1 \cap S_2$ is nonempty. We call a directed graph a *digraph*, and call a directed path a *dipath*. Suppose that $G = (V,E)$ is a graph or digraph that is an input for our problem instance; then we use n to denote $|V|$, m to denote $|E|$, and, when stating running times, we assume $m = \Omega(n)$. By an *edge* we mean an arc (directed edge) of a digraph, as well as an undirected edge of a graph. For a set of nodes U and a set of edges F of a digraph, $\delta^{out}_F(U)$ denotes the set of edges in F with tail in U and head not in U, thus, $\delta^{out}_F(U) = \{(v,w) \in F : v \in U,\ w \in V - U\}$, and $d^{out}_F(U)$ denotes the size of this set, $|\delta^{out}_F(U)|$; $\delta^{in}_F(U)$ and $d^{in}_F(U)$ are defined similarly. Given two vertices s,t, an s,t *dipath* means a dipath with start-vertex s and end-vertex t. Given a digraph $G = (V,E)$ and two sets of vertices $S, T \subseteq V$, we use (S,T) *connectivity* to mean the minimum over all pairs $s \in S, t \in T$ of the maximum number of edge-disjoint s,t dipaths; by Menger's theorem, this equals $\min_{s\in S, t\in T}\{|F| : F \subseteq E,\ G - F \text{ has no } s,t \text{ dipath}\}$. We say that the digraph is (S,T) connected (or, has (S,T) connectivity of one) if there exists an s,t dipath for each pair of vertices $s \in S, t \in T$. Similarly, we say that the digraph is k-(S,T) connected if it has k edge-disjoint s,t dipaths for each pair $s \in S, t \in T$. We assume that the sets S and T are disjoint in Sections 3–7; this is without loss of generality, see Proposition 10.

By a T,S *dipath* we mean a dipath that has its start-vertex in T and its end-vertex in S.

Fact 7. *In the relaxed version of (S,T) connectivity (where all augmenting edges have heads in T), G has a T,S dipath iff the initial digraph G_0 has a T,S dipath.*

Proof. Consider any T,S dipath P of G. If P has no augmenting edges, then it is in G_0. Otherwise, consider the suffix of P between the last augmenting edge and the end-vertex of P. This subpath has start vertex in T, end vertex in S, and no augmenting edges, so it is a T,S dipath of G_0. □

The standard version (where all augmenting edges have tails in S and heads in T) is a special case of the relaxed version, hence, G has a T,S dipath iff the initial digraph G_0 has a T,S dipath.

A digraph $G = (V,E)$ with $T \subseteq V$ is called *acyclic on* T if there is no dicycle (i.e., connected di-Eulerian subgraph) in G that contains two distinct vertices of T. Observe that if G contains both a t,t' dipath and a t',t dipath for $t,t' \in T$, then the union of these two dipaths is a connected di-Eulerian subgraph (i.e., a dicycle). Another way to view this is via the reachability digraph on T; this is an auxiliary digraph with vertex set T, and for $t,t' \in T$, $t \neq t'$, it has an edge (t,t') iff G has a t,t' dipath; observe that G is acyclic on T iff the reachability digraph on T has no dicycles.

Let r be a vertex. An *in-tree (or in-branching) J^{in} rooted at r* is a digraph that has a v,r dipath for each vertex $v \in V(J^{in})$. An *out-tree (or out-branching) J^{out} rooted at r* is a digraph that has an r,v dipath for each vertex $v \in V(J^{out})$. A *directed Steiner tree* rooted at r is a digraph that has an r,t dipath for

every vertex $t \in T$, where $T \subseteq V$ is a given set of *terminal* vertices; the digraph may contain vertices of $V - \{r\} - T$; these are called the Steiner vertices or the optional vertices. In the *directed Steiner tree* problem, we are given a digraph $G = (V, E)$, nonnegative costs on the edges, $r \in V$, and $T \subseteq V$; the goal is to find a directed Steiner tree of minimum cost. The problem has a hardness threshold of $\Omega(\log^{2-\epsilon} n)$ assuming that NP is not contained in $\mathrm{ZPTIME}(n^{\mathrm{polylog}\ n})$, [19].

Theorem 8 ([2, 8, 13, 14])**.** *There exists a polynomial-time algorithm that finds a minimum cost in-branching (respectively, out-branching).*

Theorem 9 ([3, 20])**.** *There exists an $O(\log^3 n)$-approximation algorithm for the directed Steiner tree problem that runs in quasi-polynomial time.*

In the *directed Steiner network* problem, also known as the directed Steiner forest problem, we are given a digraph $G = (V, E)$, nonnegative costs on the edges, and a set of requirement pairs $D \subseteq V \times V$; the goal is to find a subgraph (V, F) of minimum cost that contains an s_i, t_i dipath for each requirement pair $(s_i, t_i) \in D$. This problem is at least as hard for approximation as the label-cover problem, see [1], [33, Corollary 16.39]; in particular, assuming that NP is not contained in $\mathrm{DTIME}(n^{\mathrm{polylog}\ n})$, the problem has a hardness threshold of $2^{\log^{(1-\epsilon)} n}$, for any fixed $\epsilon > 0$.

In the *set covering* problem, denoted SCP, we are given a ground-set U of so-called points, subsets $S_1, \ldots, S_q$ of U, and a nonnegative cost for each subset S_j, $j = 1, \ldots, q$; the goal is to cover U by picking a family of subsets from $S_1, \ldots, S_q$ of minimum cost, that is, each point of U should be in at least one of the picked subsets. A greedy algorithm achieves an approximation guarantee of $O(\log \max_{j=1}^{q} |S_j|)$, and improving on the hardness threshold of $\Omega(\log |U|)$ would imply that $\mathrm{P} = \mathrm{NP}$, see [31, 33, 10].

2.1 Basic results

This subsection has some basic results on the k-(S, T) connectivity problem.

Proposition 10. *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 a pair of new edges (v^-, v^+) and (v^+, v^-) with zero cost. For all old edges having v as tail (respectively, head), we change their tail (respectively, head) to v^+ (respectively, v^-). Finally, for each vertex $v \in S \cap T$, we replace v by v^+ in S and we replace v by v^- in T. It can be seen that this preserves any restrictions on the augmenting edges, that is, an edge has its head in T (respectively, has its tail in S) iff the old edge corresponding to it has its head in T (respectively, has its tail in S). The reduction is illustrated in Figure 2.

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^-, 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 a dipath in the original one by replacing subpaths v^-, v^+ or v^+, v^- (or v^+ or v^-) by a single vertex v. Hence, a solution to the original instance is feasible iff its corresponding solution is feasible to the transformed instance. Moreover, optimal solutions of both instances have the same cost. □

The proofs of our hardness-of-approximation results on different versions of the (S, T) connectivity problem are given below, after re-stating the relevant theorem.

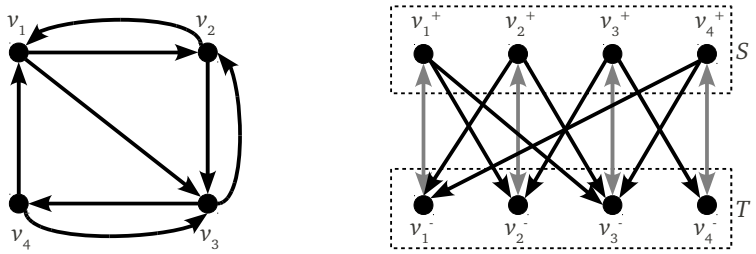


Figure 2: The reduction from an instance of the (S,T) connectivity problem where $S \cap T \neq \emptyset$ to an instance such that $S \cap T = \emptyset$. The left figure shows the original instance where $S = T = V$, and the right figure shows the transformed instance where $S \cap T = \emptyset$. In fact, the original instance is an instance of the SCSS problem. The black lines denote positive-cost edges, and the grey lines denote zero-cost edges.

Theorem 1. *Consider the k-(S,T) connectivity problem with the connectivity parameter equal to one. The hardness of approximation of the problem depends on the version of the problem (and thus on the restrictions on the augmenting edges).*

- *The standard (S,T) connectivity problem is at least as hard as the SCSS problem.*
- *The relaxed (S,T) connectivity problem is at least as hard as the directed Steiner tree problem.*
- *The (S,T) connectivity problem (without any restrictions on the augmenting edges) is at least as hard as the directed Steiner network problem.*

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

The standard (S,T) connectivity problem: The reduction from the SCSS problem to the standard (S,T) connectivity problem is straightforward. In fact, the SCSS problem is a special case of the standard (S,T) connectivity problem where $S = T = V$. It is clear that the restriction on heads and tails of augmenting edges holds because $S = T = V$. Moreover, By Proposition 10, 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 consists of a digraph $G' = (V', E')$ with non-negative cost on edges, a root vertex $r \in V'$ and a set of terminals $S' \subseteq V'$. We may assume that $r \notin S'$. Moreover, we may assume that each terminal $s \in S'$ is incident to a unique edge which is outgoing from s and has zero-cost. Otherwise, we can replace each terminal $s \in S'$ by a dummy terminal s^+ and attach s^+ to s by a zero-cost edge (s^+, s). Observe that the reduction does not increase the cost or violate the feasibility of an optimal solution. We construct the digraph G for the instance of the relaxed (S,T) connectivity problem by starting with G', and then adding auxiliary edges with zero-cost from the root vertex r to all non-terminal vertices. Thus, the digraph is $G = (V, E_0 \cup E)$, where $V = V'$ and

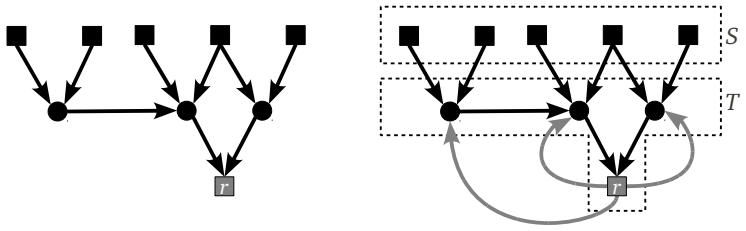


Figure 3: The reduction from an instance of the directed Steiner tree problem to an instance of the relaxed (S, T) connectivity problem. The left figure shows the instance of the former problem. The squares denote terminals, and the circles denote Steiner vertices; r is the root vertex. The right figure shows the instance of the relaxed (S, T) connectivity problem. The black lines denote positive cost edges, and the grey lines denote zero-cost edges.

$E_0 \cup E = E' \cup \{(r, v) : v \in V' - S'\}$. We define S to be the set of terminals S', and T to be the set of non-terminal vertices, that is, $S = S'$ 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 3.

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 any solution of the directed Steiner tree problem to a solution of the relaxed (S, T) connectivity problem by adding all of the auxiliary edges. Conversely, we can transform any solution of the relaxed (S, T) connectivity problem to a solution of the directed Steiner tree problem by removing all auxiliary edges. Thus, the reduction is approximation-preserving.

The (general) (S, T) connectivity problem: The reduction from the directed Steiner network problem is as follows. The given instance of the directed Steiner network problem consists of a digraph $G' = (V', E')$ with non-negative cost on the edges, and a set of requirement pairs $D \subseteq V' \times V'$. We may assume that there exist a set of sources $S \subseteq V'$ and a set of sinks $T \subseteq V'$, such that $D \subseteq S \times T$. 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 requirement pair (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 the cost or violate the feasibility. Note that each source and sink may occur in more than one requirement pair, e.g., we may have both (s, t_1) and (s, t_2) in D. We construct the digraph G for the instance of the (general) (S, T) connectivity problem by starting with G', and then adding some auxiliary edges with zero cost. 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) to G with zero cost. In other words, we pad the digraph with auxiliary edges to handle the new

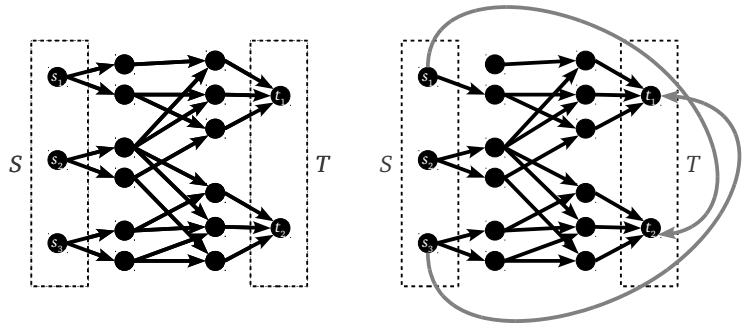


Figure 4: The reduction from an instance of the directed Steiner network problem to an instance of the (general) (S,T) connectivity problem. The left figure shows the instance of the former problem. The set of requirement pairs in this instance is $\{(s_1,t_1),(s_2,t_1),(s_2,t_2),(s_3,t_2)\}$. The right figure shows the instance of the (general) (S,T) connectivity problem. The black lines denote positive-cost edges, and the grey lines denote zero-cost edges.

requirement pairs implicit in the (general) (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. The construction is valid for the (general) (S,T) connectivity problem because there is no restriction on augmenting edges. The reduction is illustrated in Figure 4.

Given a solution to the directed Steiner network problem, we can transform it to a solution to the (general) (S,T) connectivity problem by adding all auxiliary edges. Conversely, given a solution to the (general) (S,T) connectivity problem, we can transform it to a solution to the directed Steiner network problem by removing all auxiliary edges. In other words, if $G''=(V',F')$ denotes a solution to the directed Steiner network problem, and A denotes the set of auxiliary edges, then G'' has an s,t dipath for all requirement pairs $(s,t)\in D$ iff $(V,F'\cup A)$ has an s,t dipath for all $s\in S$, $t\in T$; this holds because an auxiliary edge (s,t) is in no S,T dipath except the dipath from s to t given by the edge. The cost of the two solutions are the same because all auxiliary edges have zero-cost. Thus, the reduction is approximation-preserving. □

2.2 Frank's results on rooted connectivity

We discussed Frank's results on min-cost rooted out-subgraphs in Section 1.2; the results also apply for min-cost rooted in-subgraphs. We have stated both versions below. First, we state a result for augmenting rooted connectivity by one. Recall that a digraph $G'=(V',E')$ is said to be k-(r,T) connected, where $k\geq 0$ is an integer, $r\in V'$ and $T\subseteq V'$, if it has k edge-disjoint r,t dipaths, for each $t\in T$.

Theorem 11 (Frank 1999 [13])**.** *Given a digraph $G=(V,E_0\cup E)$, a vertex $r\in V$, a set of vertices $T\subseteq V-\{r\}$ that contains the head of every edge in E, and positive costs for the edges in E, such that $G_0=(V,E_0)$ is k-(r,T) connected, there is an $O(m(n+m))$-time algorithm for finding a set of edges $F\subseteq E$ of minimum cost such that the subgraph $(V,E_0\cup F)$ is $(k+1)$-(r,T) connected.*

Similarly, given G, $r\in V$, and positive costs on E as above, and a set of vertices $S\subseteq V-\{r\}$ that contains the tail of every edge in E, such that $G_0=(V,E_0)$ is k-(S,r) connected, there is an $O(m(n+m))$-time algorithm for finding a set of edges $F\subseteq E$ of minimum cost such that the subgraph $(V,E_0\cup F)$ is

$(k+1)$-(S,r) *connected.*

Next, we state a result of Frank on a version of the min-cost k-(r,T) connectivity problem: there is an LP relaxation that is integral, i.e., the problem can be captured as a linear programming problem. Our algorithmic results in Section 5 rely on Frank's results on rooted k-connectivity. Also, our algorithms in Section 4 apply Frank's results with $k = 2$.

Theorem 12 (Frank 2009 Theorems 4.4,5.9[14])**.** *Given a digraph $G = (V, E_0 \cup E)$, a vertex $r \in V$, a set of vertices $T \subseteq V - \{r\}$ that contains the head of every edge in E, and positive costs on the edges in E, there is a polynomial-time algorithm for finding a set of edges $F \subseteq E$ of minimum cost such that the subgraph $(V, E_0 \cup F)$ is k-(r,T) connected. Moreover, the optimal cost equals the optimal value of a linear-programming (LP) relaxation.*

Similarly, given G, $r \in V$, and positive costs on E as above, and a set of vertices $S \subseteq V - \{r\}$ that contains the tail of every edge in E, there is a polynomial-time algorithm for finding a set of edges $F \subseteq E$ of minimum cost such that the subgraph $(V, E_0 \cup F)$ is k-(S,r) connected. Moreover, the optimal cost equals the optimal value of a linear-programming (LP) relaxation.

2.3 Reducing algorithm for digraphs

This subsection presents a simple algorithm that given a digraph and a set of its vertices Z, finds a minimal subset of Z that preserves some reachability properties; precise statements are given in the next result. We apply this algorithm in Sections 3 and 4.

Proposition 13. *Let $H' = (V', E')$ be a digraph, and let $Z \subseteq V'$ be a set of vertices. Then there is a polynomial-time algorithm to find a subset Y of Z such that*

(1) for each vertex $v \in Z$ there is a vertex $y \in Y$ such that H' has a v, y dipath,

(2) H' has no dipath from any vertex of Y to another vertex of Y, and

(3) if H' has a dipath from a vertex $y \in Y$ to a vertex $v \in Z$, then y and v are in the same strongly-connected component of H'.

Moreover, the algorithm can be implemented to run in time $O(|Y| \cdot (|V'| + |E'|))$.

Proof. We obtain a running time of $O(|Y| \cdot (|V'| + |E'|))$ by applying the following method to output the vertices of Y sequentially, in linear running time per vertex.

We contract the strongly-connected components of H' into single vertices, to obtain an acyclic digraph H''. Observe that Y can have at most one vertex from each strongly-connected component of H'. Thus we transform the problem from H' to the acyclic digraph H'' in an obvious way. (A vertex v of H'' is in Z iff the strongly-connected component of H' associated with v contains a vertex of Z; similarly, a set Y of H'' can be mapped to a set Y of H' of the same size.) Then we assign a topological numbering to the vertices of H''. Moreover, we ensure that Z has at most one vertex from each strongly-connected component of H'.

We start with $X = Z$. We output the vertex y of H'' that is in X and has the highest topological number; thus this vertex is placed in Y. Then we find the vertices of X that have dipaths to y in H''; we remove all these vertices (including y) from X. We repeat this step until X becomes empty. Clearly, each iteration (of adding a vertex to Y) updates X and runs in linear time, but H'' and its topological numbering are not updated.

To verify the correctness, note that the initial set X satisfies

($*$) for each vertex $v \in Z - X$ there is a vertex $y \in Y$ such that H' has a v, y dipath.

Whenever we add a vertex y to Y, we remove from X all vertices v that can reach y. Clearly, this preserves ($*$). At termination, $X = \emptyset$ and ($*$) holds, hence, (1) holds. Consider (2): whenever we add a vertex y to Y, there is no dipath from y to vertices already in Y, and there is no dipath from y to vertices in the current set X; hence, (2) holds. Consider (3): whenever we add a vertex y to Y, then y has the highest topological number among the vertices in the current X; this implies that (3) holds. (If (3) fails, then there is a $y \in Y$ and a $v \in Z$ such that v and y are in different strongly-connected components of H', and there is a y, v dipath in H'. Thus, y and v are associated with distinct vertices of H'', call them y'' and v'', and H'' has a dipath from y'' to v''. This is not possible, because when we add y to Y, then either $v \in X$ or $v \notin X$; in the first case, v has a higher topological number than y, and in the second case, y and v would have been removed from X at the same step.) □

3 A 2-approximation algorithm for standard (S, T) connectivity

This section has our 2-approximation algorithm for the standard version of the (S, T) connectivity problem, that is, the problem of finding an (S, T) connected digraph of minimum cost, assuming that each augmenting edge has its tail in S and its head in T.

This problem is a generalization of the SCSS problem. We sketch a well-known 2-approximation algorithm for the latter problem, see [17]: Let opt denote the cost of an optimal solution. The algorithm picks any vertex to be the root vertex r, and then computes a min-cost out-branching (V, F^{out}) with root r; similarly, the algorithm computes a min-cost in-branching (V, F^{out}) with root r; then, the algorithm outputs $(V, F^{out} \cup F^{in})$. It can be seen that the solution is strongly-connected. Moreover, it can be seen that the cost of the solution is $\leq 2\mathrm{opt}$.

Recall that a T, S dipath is a dipath that has its start-vertex in T and its end-vertex in S; moreover, G has a T, S dipath iff the initial digraph G_0 has one.

We consider two cases:

1. G has a T, S dipath.
2. G has no T, S dipath.

We give a 2-approximation algorithm for the first case, by designing a simple extension of the above 2-approximation algorithm for the SCSS problem. We handle the second case by giving a polynomial-time algorithm that solves it optimally.

3.1 A 2-approximation algorithm for the case when a T, S dipath exists

First, suppose that there exists a T, S dipath, call it $\widehat{P}$, with start-vertex $\widehat{t} \in T$ and end-vertex $\widehat{s} \in S$. Then, we apply Frank's algorithm (see Theorem 11) for min-cost rooted in-subgraphs to our digraph $G = (V, E_0 \cup E)$ with root $\widehat{t}$, to find a min-cost set of edges F^{in} such that the rooted in-subgraph $(V, E_0 \cup F^{in})$ is $(S, \widehat{t})$ connected, that is, the subgraph contains an $s, \widehat{t}$ dipath for each vertex $s \in S$. Next, we apply Frank's algorithm (see Theorem 11) for min-cost rooted out-subgraphs to our digraph $G = (V, E_0 \cup E)$, but now we take the root to be $\widehat{s}$ and we find a min-cost set of edges F^{out} such that the "out-subgraph" $(V, E_0 \cup F^{out})$ is $(\widehat{s}, T)$ connected, that is, the subgraph contains an $\widehat{s}, t$ dipath for each vertex $t \in T$. The algorithm outputs $F^{in} \cup F^{out}$ as its solution. Below, we show that the algorithm is correct, that is, the

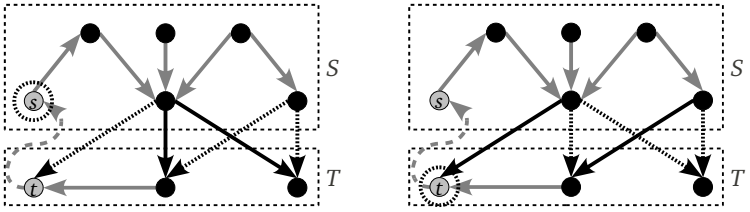


Figure 5: The figure illustrates the working of our 2-approximation algorithm for the standard (S,T) connectivity problem on an example that has a T,S dipath. The left figure shows an instance of the rooted out-subgraph problem with root vertex s and set of terminals T. The right figure shows an instance of the rooted in-subgraph problem with root vertex t and set of terminals S. The grey lines denote the edges of the initial digraph G_0. The grey dash-lines denote the T,S dipath. The black lines (solid and dash) denote the augmenting edges. The black solid-lines denote the augmenting edges chosen by the algorithm.

subgraph $\widehat{G} = (V, E_0 \cup F^{in} \cup F^{out})$ is (S,T) connected, and it achieves an approximation guarantee of 2, that is, $c(F^{in} \cup F^{out}) \leq 2\mathrm{opt}$.

Proposition 14. *Suppose that the digraph G has a T,S dipath. Then the above algorithm runs in polynomial time, and finds a feasible solution of cost $\leq 2\mathrm{opt}$.*

Proof. First, we show that the output is correct, that is, the digraph $\widehat{G}$ returned by the algorithm is (S,T) connected. Consider any pair of vertices s,t where $s \in S$ and $t \in T$. Observe that $\widehat{G}$ is $(S,\widehat{t})$ connected; similarly, $\widehat{G}$ is $(\widehat{s},T)$ connected. Hence, $\widehat{G}$ has an s,t dipath of the form

$$s \to \ldots (F^{in}) \ldots \to \widehat{t} \to \ldots (\widehat{P}) \ldots \to \widehat{s} \to \ldots (F^{out}) \ldots \to t.$$

Therefore, $\widehat{G}$ is (S,T) connected.

To see that the algorithm achieves an approximation guarantee of 2, note that a set of edges that is feasible to the (S,T) connectivity problem is also feasible to each of the two rooted subproblems solved by Frank's algorithm, hence, the cost of a feasible solution to each rooted subproblem is $\leq \mathrm{opt}$. Therefore, $\widehat{G}$ has cost $\leq 2\mathrm{opt}$.

The algorithm runs in polynomial time, because a T,S dipath of G_0 can be found in linear time (if it exists), and each of the two applications of Frank's algorithm runs in polynomial time (see Theorem 11). This completes the proof. □

Figure 5 illustrates the working of this algorithm.

3.2 A 1-approximation algorithm for the case of no T,S dipath

Recall the reducing algorithm from Section 2.3, and the conditions (1), (2), and (3) from Proposition 13. We apply that algorithm to the initial digraph G_0 and we take $Z = S$. We take the output $Y \subseteq Z$ to be $\widetilde{S} \subseteq S$. Then conditions (1), (2), and (3) apply to $\widetilde{S}$ and S. (Thus (1) for each $s \in S$ there is an $s' \in \widetilde{S}$ such that G_0 has an s,s' dipath; (2) G_0 has no dipath between two vertices of $\widetilde{S}$, and (3) if G_0 has a dipath from $s' \in \widetilde{S}$ to $s \in S$, then both s,s' are in the same strongly-connected component of G_0.) After that, for each vertex $s_i \in \widetilde{S}$, we apply Frank's "rooted out-subgraph" algorithm to the original digraph, taking the root to

be s_i, and we compute an augmenting edge set $F^{out}(s_i)$ of minimum cost such that $(V, E_0 \cup F^{out}(s_i))$ is (s_i, T) connected.

Proposition 15. *Suppose that the digraph G has no T, S dipath. Then the above algorithm finds an optimal solution to the (S, T) connectivity problem. The algorithm runs in time $O(|\widetilde{S}|m(n+m)) = O(nm^2)$.*

Proof. Let $\widehat{G}$ denote the digraph returned by the algorithm, that is, $\widehat{G} = (V, E_0 \bigcup_{s_i \in \widetilde{S}} F^{out}(s_i))$. First, we show that $\widehat{G}$ is (S, T) connected. Consider any vertex $s \in S$. Then the set $\widetilde{S}$ found by the reducing algorithm has a vertex s' such that G_0 has a dipath from s to s', by (1) in Proposition 13. Moreover, each vertex in $\widetilde{S}$ has been chosen as the root vertex for an application of Frank's algorithm, hence, G_0 together with the augmenting edge set added via Frank's algorithm contains an s', t' dipath, for all $t' \in T$. Thus, $\widehat{G}$ has an s, t' dipath, for all $t' \in T$.

Next, we show that the cost of $\widehat{G}$ is $\leq$ opt. Consider an optimal set of augmenting edges E^*, and an arbitrary vertex $s \in \widetilde{S}$. $G^* = G_0 + E^*$ must have an s, t' dipath from s to each $t' \in T$. Let $E^*(s)$ denote the subset of the augmenting edges in E^* that are used by these dipaths; the tail vertex of each of these augmenting edges must be reachable from s. Then, by Proposition 13, the tail vertices of these augmenting edges are in the same strongly-connected component as s (by (3) in the proposition); moreover, for any other vertex $s' \in \widetilde{S}$, there is no dipath from s' to any vertex in the strongly-connected component of s (by (2) in the proposition). Thus the sets $\{E^*(s) : s \in \widetilde{S}\}$ form a subpartition of E^*, i.e., each augmenting edge is in at most one of the sets $E^*(s)$. Finally, note that $E^*(s)$ forms a feasible solution for our application of Frank's "rooted out-subgraph" algorithm with root s, hence, the cost of the augmenting edges for this application is $\leq c(E^*(s))$. Summing over the cost of the augmenting edges for all of the applications of Frank's "rooted out-subgraph" algorithm with roots in $\widetilde{S}$, we see that the total cost is $\leq \sum_{s \in \widetilde{S}} c(E^*(s)) \leq$ opt.

The bound on the running time follows from the fact that the reducing algorithm runs in linear time per vertex of $\widetilde{S}$, and each application of Frank's algorithm runs in time $O(m(n+m))$. □

3.3 Combining the cases of standard (S, T) connectivity

To solve the standard (S, T) connectivity problem (without any further assumptions), we first run a depth first search algorithm to find a T, S dipath, if one exists. If there is no such dipath, then we run the algorithm given in Section 3.2. Otherwise, we run the 2-approximation algorithm given in Section 3.1. Combining these two cases, we get a 2-approximation algorithm for the problem.

Theorem 2. *Consider the standard version of the (S, T) connectivity problem; thus $k = 1$. There is a 2-approximation algorithm that runs in polynomial time.*

4 A 4-approximation algorithm for standard 2-(S, T) connectivity

We say that a digraph is 2-(S, T) *connected* if it contains two edge-disjoint s, t dipaths for every vertex $s \in S$ and every vertex $t \in T$. In this section, we give a 4-approximation algorithm for the standard version of the *minimum-cost* 2-(S, T) *connected digraph* problem, or, in brief, the standard 2-(S, T) *connectivity* problem. Recall that the standard version has $E \subseteq S \times T$, that is, each augmenting edge has its tail in S and its head in T.

Many of the steps of the algorithm and analysis occur in "symmetric pairs," for example, we may apply some procedure to a vertex in S, and then we may apply a similar procedure to a vertex in T. We describe one of these procedure in detail, but not the second one.

4.1 Preliminaries for the approximation algorithm

Let P be a dipath of G. For vertices v_i, v_j of P, we denote by $P(v_i, v_j)$ the subpath of P that starts with v_i and ends with v_j. Similarly, for edges e_i, e_j of P, we denote by $P(e_i, e_j)$ the subpath of P that starts with e_i and ends with e_j.

Recall that a T, S dipath is a dipath whose start-vertex is in T and whose end-vertex is in S. Let ν denote the maximum number of edge-disjoint T, S dipaths of G. Observe that ν is the same for G and for the initial digraph G_0. (To see this, consider a set of ν edge-disjoint T, S dipaths of G, and for each chosen T, S dipath P, replace it by the suffix of P following the last augmenting edge of P; this gives a set of ν edge-disjoint T, S dipaths of G_0.) Usually, we will consider G_0 whenever we discuss T, S dipaths. Our algorithm has three cases depending on whether ν is zero, one, or two. The case of $\nu = 1$ appears to be substantially more difficult than the other two cases.

The algorithm starts with the (edge set of the) initial digraph $G_0 = (V, E_0)$, and adds augmenting edges in several steps. We denote the current digraph by $\widehat{G}$ and the current set of augmenting edges by $\widehat{E}$. Thus, $\widehat{G} = (V, E_0 \cup \widehat{E})$, and initially, $\widehat{G} = G_0$, $\widehat{E} = \emptyset$.

4.2 No T, S dipaths

First, suppose that $\nu = 0$, i.e., G (and G_0) have no T, S dipaths. We focus on the initial digraph G_0.

The key subroutine for our algorithm is an extension of Frank's algorithm for the min-cost "rooted out-subgraph" problem. Frank gave polynomial-time algorithms and min-max theorems for the special case of the k-(S, T) connectivity problem where S consists of a single vertex and every augmenting edge has its head in T, see Theorem 12, and [13, 14]. Thus, given a root vertex r and assuming $E \subseteq V \times T$, Frank's algorithm computes a minimum-cost set of augmenting edges such that the resulting digraph is k-(r, T) connected.

Recall the reducing algorithm from Section 2.3, and the conditions (1), (2), and (3) from Proposition 13. We apply that algorithm to G_0 and we take $Z = S$. We take the output $Y \subseteq Z$ to be $\widetilde{S} \subseteq S$. Then conditions (1), (2), and (3) apply to $\widetilde{S}$ and S. After that, for each vertex $s_i \in \widetilde{S}$, we apply Frank's "rooted 2-outconnected-subgraph" algorithm to the original digraph, taking the root to be s_i, and we compute an augmenting edge set $F^{out}(s_i)$ of minimum cost such that $(V, E_0 \cup F^{out}(s_i))$ is 2-(s_i, T) connected.

Similarly, we apply the reducing algorithm (in Section 2.3) to the initial digraph G_0, to find a subset $\widetilde{T}$ of T such that conditions (1), (2) and (3) of Proposition 13 hold with $\widetilde{T} = Y$ and $T = Z$. (Formally speaking, we apply the algorithm to the digraph obtained from G_0 by replacing each edge (v, w) by the reverse edge (w, v). Thus (1) for each $t \in T$ there is a $t' \in \widetilde{T}$ such that G_0 has a t', t dipath; (2) G_0 has no dipath between two vertices of $\widetilde{T}$, and (3) if G_0 has a dipath from $t \in T$ to $t' \in \widetilde{T}$, then both t, t' are in the same strongly-connected component of G_0.) Then, for each vertex $t_j \in \widetilde{T}$, we apply Frank's "rooted 2-inconnected-subgraph" algorithm to the original digraph, taking the root to be t_j, and we compute an augmenting edge set $F^{in}(t_j)$ of minimum cost such that $(V, E_0 \cup F^{in}(t_j))$ is 2-(S, t_j) connected.

All these augmenting edge sets are added to the current digraph; thus we have $\widehat{E} = \bigcup_{s_i \in \widetilde{S}} F^{out}(s_i) \bigcup_{t_j \in \widetilde{T}} F^{in}(t_j)$.

Lemma 16. *The current digraph $\widehat{G}$ is 2-(S, T) connected.*

Proof. To see that $\widehat{G}$ is 2-(S, T) connected, consider any cut $(U, V - U)$ with $U \cap S \neq \emptyset$ and $(V - U) \cap T \neq \emptyset$. If any vertex s_i of $\widetilde{S}$ is in U, then the cut has ≥ 2 edges (since $\widehat{G}$ is 2-(s_i, T) connected, $\forall s_i \in \widetilde{S}$). Similarly, if any vertex t_j of $\widetilde{T}$ is in $V - U$, then the cut has ≥ 2 edges. In the remaining case, we have

$\widetilde{T} \subseteq U$ and $\widetilde{S} \subseteq V - U$. Let s be a vertex of $U \cap S$ and let t be a vertex of $(V - U) \cap T$; both s, t exist by our choice of U. Moreover, by Proposition 13, there is a dipath P_s from s to some vertex $s' \in \widetilde{S}$ (by (1) in the proposition). Similarly, by Proposition 13, there is a dipath P_t from some vertex $t' \in \widetilde{T}$ to t. The dipaths P_s and P_t have no vertex (or edge) in common; otherwise, their union would contain a T, S dipath. It follows that the cut $(U, V - U)$ has at least two edges, one from P_s and one from P_t. This completes the proof. □

The next lemma shows that the cost of the solution digraph $\widehat{G}$ is $\leq \mathrm{opt}$. The proof is similar to the proof of Proposition 15.

Lemma 17. *The cost of the current digraph $\widehat{G}$ is $\leq 2\mathrm{opt}$.*

Proof. The key point is that the total cost of the edges in $\bigcup_{s \in \widetilde{S}} F^{out}(s)$ is $\leq \mathrm{opt}$; similarly, the total cost of the edges in $\bigcup_{t \in \widetilde{T}} F^{in}(t)$ is $\leq \mathrm{opt}$.

Consider the first claim. Consider an optimal set of augmenting edges E^*, and an arbitrary vertex $s \in \widetilde{S}$; let $E^*(s)$ denote the subset of the augmenting edges in E^* that are used by the two edge-disjoint s, t' dipaths from s to each $t' \in T$. Arguing as in the proof of Proposition 15, it can be seen that the sets $\{E^*(s) : s \in \widetilde{S}\}$ form a subpartition of E^*. Moreover, $E^*(s)$ forms a feasible solution for our application of Frank's "rooted 2-outconnected-subgraph" algorithm with root s, hence, the cost of the augmenting edges for this application is $\leq c(E^*(s))$. Summing over the cost of the augmenting edges for all of the applications of Frank's algorithm with roots in $\widetilde{S}$, we see that the total cost is $\leq \sum_{s \in \widetilde{S}} c(E^*(s)) \leq \mathrm{opt}$. □

4.3 Two edge-disjoint T, S dipaths

Suppose that G has two edge-disjoint T, S dipaths. We consider G_0, since the maximum number of edge-disjoint T, S dipaths is the same in G and in G_0. We find two edge-disjoint T, S dipaths in G_0, call them P_1 and P_2; this can be done via an application of a maximum s, t flow algorithm to find an integral flow, with T as the set of sources and S as the set of sinks. Let t_1, s_1 be the start-vertex and end-vertex of P_1, and let t_2, s_2 be the start-vertex and end-vertex of P_2.

Then, for $i = 1, 2$, we apply Frank's "rooted 2-outconnected-subgraph" algorithm to the original digraph, taking the root to be s_i, and we compute an augmenting edge set $F^{out}(s_i)$ of minimum cost such that $(V, E_0 \cup F^{out}(s_i))$ is 2-(s_i, T) connected. Clearly, the cost of each of these augmenting edge sets is $\leq \mathrm{opt}$.

Similarly, for $j = 1, 2$, we apply Frank's "rooted 2-inconnected-subgraph" algorithm to the original digraph, taking the root to be t_j, and we compute an augmenting edge set $F^{in}(t_j)$ of minimum cost such that $(V, E_0 \cup F^{in}(t_j))$ is 2-(S, t_j) connected.

We add all these augmenting edge sets to the current digraph, i.e., we let $\widehat{E} = E_0 \bigcup F^{out}(s_1) \bigcup F^{out}(s_2) \bigcup F^{in}(t_1) \bigcup F^{in}(t_2)$.

Lemma 18. *The current digraph $\widehat{G} = G_0 + \widehat{E}$ is 2-(S, T) connected, and the cost of $\widehat{E}$ is $\leq 4\mathrm{opt}$.*

Proof. Clearly, $c(\widehat{E}) \leq 4\mathrm{opt}$, since each of the four sets of augmenting edges added to $\widehat{E}$ has cost $\leq \mathrm{opt}$.

To see that $\widehat{G}$ is 2-(S, T) connected, consider any cut $(U, V - U)$ with $U \cap S \neq \emptyset$ and $(V - U) \cap T \neq \emptyset$. If s_1 or s_2 is in U, then the cut has ≥ 2 edges (since $\widehat{G}$ is 2-(s_i, T) connected for $i = 1, 2$). Similarly, if t_1 or t_2 is in $V - U$, then the cut has ≥ 2 edges. In the remaining case, we have $t_1, t_2 \in U$ and $s_1, s_2 \in V - U$; then the cut has ≥ 1 edge from each of P_1 and P_2. This completes the proof. □

4.4 One (but not two) edge-disjoint T, S dipaths

Recall that the maximum number of edge-disjoint T, S dipaths is the same in G and in G_0. Suppose that G_0 has a T, S dipath, but it does not have two edge-disjoint T, S dipaths; thus $\nu = 1$. Then there exists a (T, S) cut of G_0 of size one. (Such a cut can be found by applying the max-flow min-cut theorem and algorithm, with T as the set of sources and S as the set of sinks.) We define a *cut edge* (of G_0, or equivalently of G) to be an edge whose deletion results in a digraph that has no T, S dipaths. (Observe that an edge e is a cut edge w.r.t. G_0 iff e is a cut edge w.r.t. G.)

We start by finding a T, S dipath of G_0 with the minimum number of edges, call it $\widehat{P}$. We denote the start-vertex and end-vertex of $\widehat{P}$ by $\widehat{t}$ and $\widehat{s}$, respectively. Clearly, every cut edge is contained in $\widehat{P}$. Let $e_1, e_2, \ldots, e_\ell$ denote all the cut edges, listed according to their order of occurrence in $\widehat{P}$.

Lemma 19. *Let P be any T, S dipath. Then all of the cut edges are in P, and their order of occurrence is the same in P and $\widehat{P}$, namely, $e_1, e_2, \ldots, e_\ell$.*

Proof. Clearly, each of the cut edges is in P. For the second part, we argue by contradiction. Let j be the smallest index (possibly, $j = 1$) such that e_j does not occur as the jth cut edge of P. Then, $P - e_j$ contains a dipath starting with e_{j-1} (starting with some $t \in T$, if $j = 1$) and ending with e_q, $q \geq j + 1$; then the union of this dipath with $\widehat{P}$ contains a T, S dipath of $\widehat{G} - e_j$. □

We use t^* to denote the tail vertex of e_1, and s^* to denote the head vertex of e_ℓ. (In general, $t^* \notin T$ and $s^* \notin S$, but it turns out that t^* and s^* have some of the properties of the vertices in T and S, respectively.) The next lemma states that the deletion of a non cut edge from G_0 cannot disconnect t^* from s^*.

Lemma 20. *Let f be an edge of $\widehat{P}(e_1, e_\ell)$ that is not a cut edge. Then $G_0 - f$ has a dipath from t^* to s^*.*

Proof. Since f is not a cut edge, $G_0 - f$ has a T, S dipath P'. Moreover, P' contains all of the cut edges, and their order of occurrence on P' is $e_1, e_2, \ldots, e_\ell$. Thus $P'(e_1, e_\ell)$ is the required dipath of $G_0 - f$ from t^* to s^*. □

We construct the digraph $\widehat{G}$ by starting with G_0, and then applying preprocessing steps to obtain a digraph that is (S, T) connected such that the deletion of any one non cut edge preserves the (S, T) connectivity; in other words, if the removal of a single edge from $\widehat{G}$ results in a digraph that is not (S, T) connected, then the removed edge must be a cut edge.

The preprocessing steps first apply Frank's "rooted 2-inconnected-subgraph" algorithm with root t^*, to compute an augmenting edge set $F^{in}(t^*)$ of minimum cost such that $(V, E_0 \cup F^{in}(t^*))$ is 2-(S, t^*) connected. Then, we apply Frank's "rooted 2-outconnected-subgraph" algorithm with root s^*, to compute an augmenting edge set $F^{out}(s^*)$ of minimum cost such that $(V, E_0 \cup F^{out}(s^*))$ is 2-(s^*, T) connected. Let $\widehat{G}$ be the digraph $G_0 + F^{in}(t^*) + F^{out}(s^*)$; note that the initial digraph is a subgraph of $\widehat{G}$.

Lemma 21. *The digraph $\widehat{G} = G_0 + F^{in}(t^*) + F^{out}(s^*)$ has cost $\leq 2\mathrm{opt}$, and moreover, it is both 2-(S, t^*) connected and 2-(s^*, T) connected.*

Proof. We claim that the digraph $G^* = G_0 + E^*$ given by an optimal solution E^* to the 2-(S, T) connectivity problem satisfies the requirements of both 2-(S, t^*) connectivity and 2-(s^*, T) connectivity. Consider the first requirement. Informally speaking, G^* is 2-(S, T) connected and G_0 has two edge disjoint dipaths from T to t^*, hence, G^* is 2-(S, t^*) connected; a rigorous proof is given below. Similarly, it can be proved that G^* is 2-(s^*, T) connected. Moreover, it can be seen that the condition required by Frank's "rooted

2-inconnected-subgraph" algorithm applies, that is, every edge of positive cost has its tail at a vertex with positive connectivity requirement, because every augmenting edge has its tail vertex in S. Therefore, Frank's algorithm finds an optimal solution to the "rooted 2-inconnected-subgraph" problem, and moreover, this solution has cost $\leq c(E^*) = \text{opt}$. Hence, $F^{in}(t^*)$ has cost $\leq$ opt. Similarly, $F^{out}(s^*)$ has cost $\leq$ opt. Thus, the total cost of $\widehat{G}$ is $\leq 2\text{opt}$.

We prove that G^* is 2-(S, t^*) connected by a contradiction argument. Suppose that the connectivity requirement does not hold for G^*. Then there is a cut $(U, V-U)$ of size < 2 such that $U \cap S$ is nonempty and $t^* \in V - U$. Since G^* is 2-(S, T) connected, we have $T \subseteq U$. Moreover, G^* has a T, t^* dipath (i.e., a dipath from a vertex of T to t^*) because G^* has the T, S dipath $\widehat{P}$ which contains t^*. Hence, the cut $(U, V-U)$ has one edge of the T, t^* dipath; let this edge be f. Then the digraph $G^* - f$ has no T, t^* dipath. Hence, $G_0 - f$ has no T, S dipath (if such a dipath existed, then it would contain the cut edge e_1 and its tail vertex t^*, thus it would contain a subpath from T to t^*). Thus f is a cut edge, say $f = e_j, j = 1 \ldots, \ell$. This gives a contradiction because $G_0 - e_j$ contains the T, t^* dipath $\widehat{P}(\widehat{t}, t^*)$ for each $j = 1, \ldots, \ell$. □

Suppose that the resulting digraph is not 2-(S, T) connected. Then, there exists an edge whose removal results in a digraph that is not (S, T) connected. The next lemma shows that any such edge must be a cut edge.

Lemma 22. *Consider the current digraph $\widehat{G} = G_0 + F^{in}(t^*) + F^{out}(s^*)$. Suppose that f is an edge such that $\widehat{G} - f$ is not (S, T) connected. Then f is a cut edge.*

Proof. Observe that $\widehat{G} - f$ has a dipath from every vertex of S to t^* (due to $F^{in}(t^*)$), and it has a dipath from s^* to every vertex of T (due to $F^{out}(s^*)$). If there is a t^*, s^* dipath in $\widehat{G} - f$ then $\widehat{G} - f$ would be (S, T) connected and we would get a contradiction. Lemma 20 shows that $G_0 - f$ has a t^*, s^* dipath, unless f is a cut edge. This completes the proof. □

4.5 Last step for $\nu = 1$: "Eliminating" all cut edges

The last part of the algorithm "eliminates" the cut edges; we examine the cut edges $e_1, e_2, \ldots, e_\ell$ and find a set of augmenting edges F'; we will prove that the digraph obtained by adding F' to $\widehat{G}$ is 2-(S, T) connected and it has cost $\leq 3\text{opt}$.

To see the key idea, consider the special case of one cut edge, that is, $\ell = 1$. We apply the algorithm for (S, T) connectivity to $\widehat{G} - e_1$ to find a set of augmenting edges F_1 such that $\widehat{G} - e_1 + F_1$ is (S, T) connected. Observe that $\widehat{G} - e_1$ has no T, S dipaths, hence, in this special case, our algorithm finds a set of augmenting edges F_1 of minimum cost. We claim that $\widehat{G} + F_1$ is 2-(S, T) connected. To prove this, suppose that there exists an edge e whose deletion results in a digraph that is not (S, T) connected. By Lemma 22, e is a cut edge of $\widehat{G}$; thus $e = e_1$. Then we get a contradiction, since $\widehat{G} - e_1 + F_1$ is (S, T) connected;

In general, for $\ell \geq 2$, we handle all of the cut edges in one step, by "reducing" the problem (of finding a set of augmenting edges F' such that $\widehat{G} + F' - e_i$ is (S, T) connected, for each $i \in \{1, \ldots, \ell\}$) to a single (S, T) connectivity problem on an auxiliary digraph G'. The auxiliary digraph G' is obtained from $\widehat{G}$ by

(i) deleting all the cut edges $e_1, \ldots, e_\ell$, and

(ii) adding an auxiliary edge (v, w) of zero cost for each pair $v \in S$, $w \in T$ such that $\widehat{G}$ has two edge-disjoint v, w dipaths.

As above, we apply the algorithm for (S, T) connectivity to G' to find a set of augmenting edges F' such that $G' + F'$ is (S, T) connected. Let A' denote the set of auxiliary edges of G' (edges present in G' but not in $\widehat{G}$). The correctness of this method follows from the following lemma.

Lemma 23. *Consider the auxiliary digraph G' and a set of augmenting edges F'. Then*

(i) *$G' + F'$ is (S,T) connected implies $\widehat{G} + F'$ is 2-(S,T) connected, and*

(ii) *$G_0 + F'$ is 2-(S,T) connected implies $G' + F'$ is (S,T) connected.*

Proof. Consider the easy part (i) first. Suppose that $G' + F'$ is (S,T) connected, but $\widehat{G} + F'$ is not 2-(S,T) connected. Then, by Lemma 22, there exists a cut edge $e \in \{e_1, \ldots, e_\ell\}$ such that $\widehat{G} + F' - e$ is not (S,T) connected. Consider any pair $s \in S$, $t \in T$. The augmented auxiliary digraph $G' + F'$ has an s,t dipath P', and P' contains none of the edges in $\{e_1, \ldots, e_\ell\}$, but P' may contain one or more of the auxiliary edges; now, observe that for each auxiliary edge $(v,w) \in A'$, $\widehat{G}$ has at least two edge-disjoint v,w dipaths, hence, $\widehat{G} - e$ has a v,w dipath $P_{v,w}$; thus, the union of $P' - A'$ and $\bigcup_{(v,w)\in A'} P_{v,w}$ contains an s,t dipath of $\widehat{G} - e$. We get the desired contradiction, since $\widehat{G} + F' - e$ is (S,T) connected.

Now, consider the other part. Suppose that $G_0 + F'$ is 2-(S,T) connected, but $G' + F'$ is not (S,T) connected.

Then there exists a pair $s \in S$, $t \in T$ such that

(a) $G_0 + F'$ has two edge-disjoint s,t dipaths, but

(b) $G' + F'$ has no s,t dipath, and

(c) $\widehat{G}$ does not have two edge-disjoint s,t dipaths (otherwise, G' would have the auxiliary edge (s,t)).

We derive a contradiction by showing that $\widehat{G}$ has two edge-disjoint s,t dipaths, if statements (a) and (b) hold.

Let P_1 and P_2 denote two edge-disjoint s,t dipaths of $G_0 + F'$. One of these dipaths (possibly, both of them) has an augmenting edge, otherwise, both dipaths would be contained in G_0 and thus in $\widehat{G}$. Observe that every dipath of $G_0 + F'$ that avoids all cut edges is contained in $G' + F'$. Since P_1 and P_2 are not contained in $G' + F'$, each of these dipaths must contain a cut edge. Moreover, neither P_1 nor P_2 has two or more augmenting edges; to see this, suppose that P_1 contains an augmenting edge (s_1, t_1) followed by another augmenting edge (s_2, t_2); then, by Lemma 19, all the cut edges $e_1, \ldots, e_\ell$ would occur in $P_1(t_1, s_2)$ between the two augmenting edges; but then P_2 (being edge disjoint from P_1) would not contain any cut edges.

Consider two cases:

(1) one of P_1, P_2, say P_1, contains augmenting edges, but the other one, P_2, contains no augmenting edges;

(2) P_1, P_2 both contain augmenting edges.

The next claim states simple but useful properties of G.

Claim 24. *Consider the digraph G and any dipath P that has an augmenting edge α and a cut edge e. Then either*
(1) the first cut edge following α (in P) is e_1, or
(2) the last cut edge preceding α (in P) is e_ℓ.

Proof. Either α precedes some cut edge, or it follows all cut edges. Suppose the first cut edge following α (in P) is e_i ($i \geq 2$); then the union of $P(\alpha, e_i)$ and $\widehat{P} - e_1$ contains a T,S dipath, a contradiction to the definition of e_1. Similarly, if the last cut edge preceding α (in P) is e_i ($i < \ell$), then we get a contradiction to the definition of e_ℓ. The claim follows. □

By way of contradiction, assume that $\widehat{G}$ does not have two edge-disjoint s,t dipaths. Then there exists an edge e such that $\widehat{G}-e$ has no s,t dipath.

First, consider case (1): P_1 has augmenting edges, but P_2 has none. If $e \notin P_2$, then we are done, because $P_2 = P_2 - e$ is an s,t dipath of $\widehat{G}$. Now, suppose $e \in P_2$. Consider P_1, and let α be its unique augmenting edge. By Claim 24, either $e_1 \in P_1$ or $e_\ell \in P_1$. First, suppose that e_1 is the first cut edge in P_1 following α; thus, the tail t^* of e_1 is in P_1. Observe that $e \notin P_1(t^*,t)$, because P_1, P_2 are edge disjoint and $e \in P_2$. Moreover, the unique augmenting edge of P_1 precedes t^*, hence, $P_1(t^*,t)$ contains no augmenting edges. Finally, observe that $\widehat{G}-e$ has a dipath P'' from s to t^*, because $\widehat{G}$ is 2-(S,t^*) connected (by Lemma 21), hence, deleting any edge results in a digraph that is 1-(S,t^*) connected, and thus has an s,t^* dipath. (We state this observation as a claim below, for further use.) It follows that $\widehat{G}-e$ contains the union of P'' and $P_1(t^*,t)$, which contains an s,t dipath.

Claim 25. *Let s and t be as above, and let e be any edge of $\widehat{G}$.*
(1) $\widehat{G}-e$ has a dipath P'' from s to t^*, where t^* denotes the tail of e_1.
(2) $\widehat{G}-e$ has a dipath P''' from s^* to t, where s^* denotes the head of e_ℓ.

Now, suppose that e_ℓ is the last cut edge in P_1 preceding α. Then, a similar argument shows that $e \notin P_1(s,s^*)$, and $P_1(s,s^*)$ contains no augmenting edges. Thus, applying Claim 25 (and its notation), $\widehat{G}-e$ contains the union of $P_1(s,s^*)$ and P''', which contains an s,t dipath.

Thus, in Case (1), we get the desired contradiction: $\widehat{G}$ has two edge-disjoint s,t dipaths.

Finally, consider Case (2): both P_1, P_2 contain augmenting edges and cut edges. Let e be an edge such that $\widehat{G}-e$ has no s,t dipath. By Claim 24, either $e_1 \in P_1$ or $e_\ell \in P_1$, and the same holds for P_2; moreover, neither P_1 nor P_2 contains two or more augmenting edges. We may fix the indices of P_1 and P_2 such that $e_1 \in P_1$ and $e_\ell \in P_2$. Then note that the dipaths $P_1(e_1,t)$ and $P_2(s,e_\ell)$ are edge disjoint, and one of them avoids e. As above, we apply Claim 25 (and its notation). If $P_1(e_1,t)$ avoids e, then the union of P'' and $P_1(e_1,t)$ is contained in $\widehat{G}-e$, and it contains an s,t dipath; otherwise, the union of P''' and $P_2(s,e_\ell)$ is contained in $\widehat{G}-e$, and it contains an s,t dipath. Thus, in Case (2), we get the desired contradiction: $\widehat{G}$ has two edge-disjoint s,t dipaths.

This completes the proof of the lemma. □

The next result summarizes the contributions of this subsection by proving the correctness and the approximation guarantee for the above algorithm.

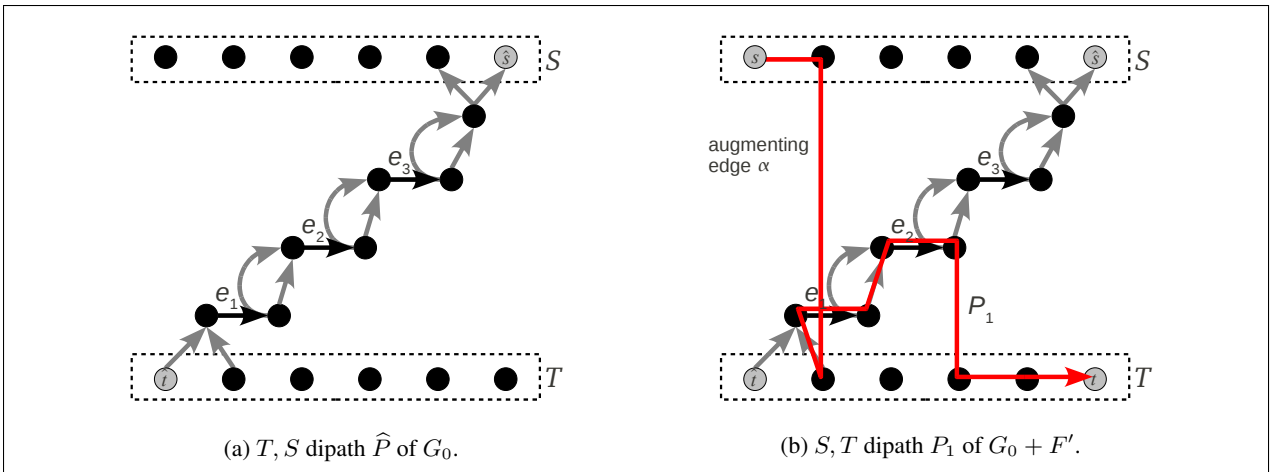
Lemma 26. *The digraph returned by the algorithm is 2-(S,T) connected, and it has cost $\leq 3\mathrm{opt}$.*

Proof. The digraph returned by the algorithm has the edge set $E_0 \bigcup F^{out}(s^*) \bigcup F^{in}(t^*) \bigcup F'$. Each of the sets of augmenting edges has cost $\leq \mathrm{opt}$, hence $\widehat{G}$ has cost $\leq 3\mathrm{opt}$.

To see the correctness, first note that $\widehat{G}$, which has the edge set $E_0 \bigcup F^{out}(s^*) \bigcup F^{in}(t^*)$, is both 2-$(S,t^*)$ connected and 2-(s^*,T) connected (see Lemma 21). Also, observe that $G_0 + E^*$ is 2-(S,T) connected, hence, by Lemma 23, $G' + E^*$ is (S,T) connected. Thus, the algorithm succeeds in finding a set of augmenting edges F' such that $G' + F'$ is (S,T) connected, and hence (by Lemma 23), $\widehat{G} + F'$ is 2-(S,T) connected. Moreover, $c(F') \leq c(E^*) = \mathrm{opt}$. □

4.6 Combining the cases of 2-(S,T) connectivity

This subsection summarizes our approximation algorithm and analysis for the min-cost 2-(S,T) connectivity problem.



(a) T, S dipath $\widehat{P}$ of G_0.

(b) S, T dipath P_1 of $G_0 + F'$.

Figure 6: The figures illustrate the notation in Lemma 23. The first figure illustrates the T, S dipath $\widehat{P}$ and its cutedges. The second figure illustrates one of the possible S, T dipaths P_1 of $G_0 + F'$.

Theorem 3. *Consider the standard version of the* 2-(S, T) *connectivity problem. There is a* 4*-approximation algorithm that runs in polynomial time.*

Proof. The correctness of the output follows from the correctness proofs of the three main cases in the algorithm.

The cost analysis follows easily from the cost analysis of the three main cases in the algorithm. In the case of no T, S dipaths, the cost of $\widehat{G}$ is $\leq 2\,\mathrm{opt}$. In the case of one T, S dipath, but not two edge-disjoint T, S dipaths, the cost of $\widehat{G}$ is $\leq 3\,\mathrm{opt}$. In the case of two edge-disjoint T, S dipaths, the cost of $\widehat{G}$ is $\leq 4\,\mathrm{opt}$. □

5 An $O(\log k \cdot \log n)$ approximation algorithm for standard k-(S, T) connectivity

5.1 Introduction to k-(S, T) connectivity

This section focuses on the standard version of the k-(S, T) connectivity problem: we are given an integer $k \geq 0$, a directed graph $G = (V, E_0 \cup E)$, two subsets S, T of V, and positive costs on the edges in E; moreover, each edge in E has its tail in S and its head in T. A digraph is called k-(S, T) connected if it has k edge-disjoint dipaths between every vertex $s \in S$ and every vertex $t \in T$. The goal is to find a subset of edges $\widehat{E} \subseteq E$ of minimum cost such that the subgraph $(V, E_0 \cup \widehat{E})$ is k-(S, T) connected.

Although the k-(S, T) connectivity problem pertains to edge-connectivity, it can be seen that the k-VCSS problem is a special case of this problem by applying the reduction in Proposition 10. In particular, the k-VCSS digraph has k openly-disjoint dipaths between every pair of vertices v, w iff the digraph resulting from the reduction has k edge-disjoint dipaths between every vertex $v^+ \in S$ and every vertex $w^- \in T$.

Thus any approximation guarantee for the min-cost k-(S, T) connected digraph problem implies the same approximation guarantee for the k-VCSS problem. We do not improve on the best known approximation guarantee for the latter problem, which is $O(\log k \cdot \log(\frac{n}{n-k}))$ [26].

The main result of this section is stated below; it is proved by generalizing the algorithm and analysis for the k-VCSS problem in [9], which in turn is based on ideas and results from [16, 22, 28, 24]. At a high level, our algorithm and analysis are almost the same as the halo-set method of [9]. But there are differences. Previous applications of the halo-set method rely on the property of "disjointness of cores," whereas our application circumvents this property. We achieve the generalization by a deeper analysis that is simpler in some ways.

Theorem 4. *There is a polynomial-time approximation algorithm for the standard version of the min-cost k-(S,T) connected digraph problem that achieves a guarantee of $O(\log k \cdot \log n)$.*

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

Our approximation algorithm and guarantee are obtained by applying the halo-set method; recall the overview from Section 1.3. First, we show that the LP-scaling technique [18] applies in our setting. Based on that, we focus on the key subproblem of increasing (S,T) connectivity by one.

Consider the following LPs: The first one, denoted $\mathrm{LP}(k)$, is a well-known LP relaxation for the k-(S,T) connectivity problem, where E_0 denotes the edge set of the initial digraph. The second one, denoted $\mathrm{LP}^{inc}(\ell)$, is a well-known LP relaxation for the problem of increasing (S,T) connectivity by one by adding edges from $E - E_\ell$ to a digraph (V, E_ℓ) that is ℓ-(S,T) connected; we may view (V, E_ℓ) as the "initial digraph."

LP for k-(S,T) connectivity

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

LP for increasing (S,T) connectivity from ℓ to $\ell+1$

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

Proposition 27. *Suppose there is an approximation algorithm for the problem of increasing the (S,T) connectivity of a digraph by one that achieves an approximation guarantee of $\beta(n)$ with respect to the LP relaxation $\mathrm{LP}^{inc}(\ell)$. Then there is an $O(\beta(n) \log k)$-approximation algorithm for the k-(S,T) connectivity problem.*

Proof. The proof follows from the well-known LP-scaling technique. The next result says that the optimal cost of a *fractional* solution to the problem of increasing (S,T) connectivity from ℓ to $\ell+1$ is $\leq 1/(k-\ell)$ times the optimal cost of a *fractional* solution to the k-(S,T) connectivity problem.

Claim 28. *Given a digraph and $\ell < k$, the optimal value of $\mathrm{LP}^{inc}(\ell)$ is $\leq 1/(k-\ell)$ times the optimal value of $\mathrm{LP}(k)$, i.e., $z^{inc} \leq z^*_k/(k-\ell)$, where z^{inc} denotes the optimal value of $\mathrm{LP}^{inc}(\ell)$ and z_k denotes the optimal value of $\mathrm{LP}(k)$.*

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

Let $E_\ell \supset E_0$ denote the set of edges in the initial digraph of the problem of increasing (S,T) connectivity from ℓ to $\ell+1$.

We construct a solution $\boldsymbol{x}'$ to $\mathrm{LP}^{inc}(\ell)$ by assigning $\boldsymbol{x}'_e = \frac{1}{(k-\ell)}$ for all edges $e \in E_k - E_\ell$, and assigning $\boldsymbol{x}'_e = 0$ otherwise. We claim that $\boldsymbol{x}'$ is a feasible solution to $\mathrm{LP}^{inc}(\ell)$. To see this, consider any set $U \subseteq V$ appearing in any constraint of $\mathrm{LP}^{inc}(\ell)$. Since $\boldsymbol{x}^*$ satisfies $\mathrm{LP}(k)$, we have $\boldsymbol{x}^*(\delta^{out}_{E_k}(U)) + d^{out}_{E_0}(U) \geq k$, hence, $\boldsymbol{x}^*(\delta^{out}_{E_k - E_\ell}(U)) + d^{out}_{E_\ell}(U) \geq \boldsymbol{x}^*(\delta^{out}_{E_k}(U)) + d^{out}_{E_0}(U) \geq k$. Moreover, we have $d^{out}_{E_\ell}(U) = \ell$, by the ℓ-(S,T) connectivity of the "initial digraph" (V, E_ℓ). Hence, we have $\boldsymbol{x}^*(\delta^{out}_{E_k - E_\ell}(U)) \geq k - \ell$. Consequently, $\boldsymbol{x}'(\delta^{out}_{E_k - E_\ell}(U)) \geq 1$. This proves our claim: $\boldsymbol{x}'$ is a feasible solution to $\mathrm{LP}^{inc}(\ell)$. Clearly, the cost of $\boldsymbol{x}'$ is $\leq 1/(k-\ell)$ times the cost of $\boldsymbol{x}^*$, hence, $z^{inc} \leq z^*_k/(k-\ell)$. □

The proof of Proposition 27 follows easily. We solve the given k-(S,T) connectivity problem by iteratively increasing the (S,T) connectivity by one via the $\beta(n)$-approximation algorithm; note that this algorithm finds a solution of cost $\leq \beta(n) z^{inc}$. Hence, the resulting set of augmenting edges has cost

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

□

In the rest of this section, we present our approximation algorithm for increasing the (S,T) connectivity by one. We assume that the initial digraph is ℓ-(S,T) connected. This assumption is valid because previous iterations of the algorithm have increased the (S,T) connectivity from zero to ℓ.

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

This subsection develops some basic results on ℓ-(S,T) connected digraphs, where ℓ is a nonnegative integer.

A *deficient set* is a set of vertices $U \subseteq V$ such that $U \cap S \neq \emptyset$, $U \cap T \neq T$, and $d^{out}(U) < \ell + 1$. Thus there exists a pair of vertices $s \in S$ and $t \in T$ such that U "separates" s and t, so any feasible solution of the $(\ell+1)$-(S,T) connectivity problem has $\geq \ell + 1$ edges in the cut $(U, V - U)$, but the current digraph has $\leq \ell$ edges in the cut. Observe that every deficient set U has $d^{out}(U) = \ell$, since we assume that the initial digraph is ℓ-(S,T) connected. The next lemma is basic and it follows from submodularity.

Lemma 29 (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. The function $d^{out}(\cdot)$ is submodular. Hence, we have

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

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

We call an inclusionwise minimal deficient set a *core*, and denote it by C, or C_i, etc. The *halo family* of a core C, denoted $\mathrm{Halo}(C)$, is the family of deficient sets containing C but containing no other cores, that is,

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

The *halo set* of C, denoted $\mathrm{H}(C)$, is the union of all members of the halo family of C, that is, $\mathrm{H}(C) = \bigcup\{W : W \in \mathrm{Halo}(C)\}$.

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

For a deficient set U, we define the *body* to be $\mathrm{Body}(U) = U \cap S$, and we define the *shadow* to be $\mathrm{Shadow}(U) = T - U$. The next lemma is a key tool for our algorithm and its analysis.

Lemma 30 (Disjointness Property)**.** *Let C and D be two distinct cores. Let U be a deficient set in $\mathrm{Halo}(C)$, and let W be a deficient set 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. Suppose that $\mathrm{Body}(U)$ and $\mathrm{Body}(W)$ intersect; otherwise, the lemma holds. For the sake of contradiction, suppose that $\mathrm{Shadow}(U)$ and $\mathrm{Shadow}(W)$ intersect. Then we have

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

Then by Lemma 29, $U \cap W$ is a deficient set, and thus it contains a core. We have a contradiction because C is the unique core contained in U, D is the unique core contained in W, and C, D are distinct. □

5.4 Computing cores

This subsection describes an efficient algorithm for computing all of the cores, via a max s, t-flow algorithm. Recall our assumption that the current digraph is ℓ-(S, T) connected.

For each pair of vertices $s \in S, t \in T$, we construct a flow network $N_{s,t}$ with s as the source vertex and t as the sink vertex, where each edge has unit capacity. Then we apply any efficient max-flow algorithm to compute a maximum s, t flow and also we compute the residual graph $\hat{N}_{s,t}$ with respect to the maximum s, t flow. If the value of the maximum flow is less than $\ell + 1$ (the required (S, T) connectivity), then let $C_{s,t}$ denote the set of vertices reachable from the source s in $\hat{N}_{s,t}$. On the other hand, if the value of the maximum flow is $\geq \ell + 1$, then there exists no deficient set (and no core) that includes s and excludes t.

Let $\mathcal{C}'$ denote $\{C_{s,t} : s \in S, t \in T\}$. In the final step, we remove from $\mathcal{C}'$ all those sets that properly contain some other set of $\mathcal{C}'$. Let $\mathcal{C}$ be the resulting family of sets. We claim that $\mathcal{C}$ is the family of all cores.

Proposition 31. *For every pair of vertices $s \in S, t \in T$, if the above algorithm finds a set $C_{s,t}$ then the set is the unique minimal deficient set that includes s and excludes t. Moreover, the algorithm finds all of the cores by computing $\mathcal{C}$.*

Proof. Observe that the (S, T) connectivity of the initial digraph is ℓ, hence, for every pair of vertices $s \in S, t \in T$, the value of a maximum s, t-flow and the capacity of a minimum s, t-cut is $\geq \ell$.

Consider a core C and any pair of vertices $s \in S \cap C$, $t \in T - C$. The core gives a minimum s, t cut of capacity ℓ. Furthermore, we have $C \subseteq C_{s,t}$; to see this, consider the residual graph of the flow network $N_{s,t}$ (w.r.t. a maximum s, t flow) and observe that every vertex of C must be reachable from s, otherwise, we would get another minimum s, t cut $\hat{C}$ such that $C \not\subseteq \hat{C}$; this would contradict the definition of a core. Also, observe that $C_{s,t} \subseteq C$ since no vertex of $V - C$ is reachable from s in the residual graph of $N_{s,t}$.

Hence, $\mathcal{C}'$ contains all of the cores. The final step cannot remove any core from $\mathcal{C}'$ since no proper subset of a core is a deficient set. Moreover, if a deficient set U has no proper subset that is deficient, then U is a core. Thus, $\mathcal{C}$ is the family of cores. □

Corollary 32. *The number of cores is at most* $|S| \cdot |T|$.

The upper bound on the number of cores in Corollary 32 is tight. Consider the following instance of the problem of increasing the (S,T)-connectivity by one. We start with a digraph $G = (V, \emptyset)$ with vertex set $V = S \cup T$, where $|S|,\ |T| \geq 1$ and S, T are disjoint. We construct an initial digraph $G_0 = (V, E_0)$ by adding ℓ edges from each vertex $s_i \in S$ to each vertex $t_j \in T$. Clearly, the initial digraph is ℓ-(S,T) connected. Observe that 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.

5.5 Covering a halo family via the Padded-outconnectivity algorithm

A key subroutine of our algorithm uses an algorithm due to Frank [14] to cover the halo family of a core. The proof of correctness (given below) follows from the correctness of Frank's algorithm and Lemma 30 (the disjointness property). The algorithm is described below; we call it the Padded-outconnectivity algorithm.

Consider a core C, and the halo family of C. To cover the halo family, we first add so-called padding edges that cover all deficient sets that are not in the chosen halo family. In particular, for *each* core $D \neq C$, we choose an arbitrary vertex $u_D \in D \cap S$ and add new edges from u_D to each vertex $v \in \mathrm{Shadow}(D)$; thus, the set of new edges for the core D is $\{(u_D, w) : w \in \mathrm{Shadow}(D)\}$; we call these edges the *padding edges*. 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 resulting digraph, with r_C as the root vertex and T as the set of terminals; the set of augmenting edges E stays the same, and the initial digraph has all the edges of the original initial digraph G_0 as well as all of the padding edges. We claim that the set of augmenting edges $F(C)$ computed by this algorithm covers the halo family of our chosen core C. Moreover, no edge of $F(C)$ covers any deficient set in the halo-family of another core $D \neq C$; this property, together with Lemma 37, allows us to bound the cost of $F(C)$ with respect to the optimal cost.

Proposition 33. *Let C be the chosen core. Then the set of augmenting edges $F(C)$ found by the Padded-outconnectivity 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 for the input instance of the algorithm; that is, $\Pi(C) = \{U \subseteq V : r_C \in U, T - U \neq \emptyset, d^{out}(U) = \ell\}$ for the padded digraph of the $(\ell+1)$-(r_C, T) connectivity problem. We claim that $\mathrm{Halo}(C) = \Pi(C)$. We first show that $\mathrm{Halo}(C) \subseteq \Pi(C)$. Consider any deficient set $U \in \mathrm{Halo}(C)$; clearly, $r_C \in C \subseteq U$, and $T - U \neq \emptyset$. Suppose $d^{out}(U) > \ell$ in the padded digraph; then there exists a padding edge (v, w) that covers U, i.e., we have $v \in \mathrm{Body}(U)$ and $w \in \mathrm{Shadow}(U)$. By the construction of padding edges, there exists a core $D \neq C$ such that $v \in \mathrm{Body}(D)$ and $w \in \mathrm{Shadow}(D)$. Then, Lemma 30 (the disjointness property) gives a contradiction, since $v \in C \cap D \cap S$ and $w \in (T - C) \cap (T - D)$, that is, the bodies of C and D intersect, and the shadows of C and D intersect.

To complete the proof, we need to show that $\Pi(C) \subseteq \mathrm{Halo}(C)$. Let U be a deficient set in $\Pi(C)$. Then no padding edges cover U. Hence, by the construction, U contains no core distinct from C. (To see this, suppose U contains a core $D \neq C$; then $\mathrm{Shadow}(U) = T - U \subseteq T - D = \mathrm{Shadow}(D)$ which implies that there exists a padding edge (u_D, v) where $u_D \in \mathrm{Body}(D) \subseteq U \cap S$ and $v \in \mathrm{Shadow}(U) \subseteq \mathrm{Shadow}(D)$.) Thus, $U \in \mathrm{Halo}(C)$, proving that $\Pi(C) \subseteq \mathrm{Halo}(C)$. □

Lemma 34. *Let C be a core, and let $F(C)$ be an (inclusionwise) minimal set of augmenting edges that covers* $\mathrm{Halo}(C)$*. Let $D \neq C$ be another core. Then no deficient set in* $\mathrm{Halo}(D)$ *is covered by an edge in $F(C)$.*

Proof. By way of contradiction, assume that a deficient set $W \in \mathrm{Halo}(D)$ is covered by some edge $e \in F(C)$. Let $e = (v, q)$. By the minimality of $F(C)$, e covers at least one deficient set $U \in \mathrm{Halo}(C)$ (otherwise, $F(C) - \{e\}$ covers $\mathrm{Halo}(C)$). Since the edge $e = (v, q)$ covers both U and W, its tail v is in $\mathrm{Body}(U) \cap \mathrm{Body}(W)$ and its head q is in $\mathrm{Shadow}(U) \cap \mathrm{Shadow}(W)$. This contradicts Lemma 30 (the disjointness property). □

5.6 Approximation algorithms for increasing (S, T) connectivity

We increase the (S, T) connectivity by iteratively adding edges of low cost to decrease the number of cores until no cores are left. We present two different algorithms that yield the same approximation guarantee and establish the following result. The first algorithm follows a sequential greedy strategy and is easier to analyse, but the second algorithm has a better running time. The sequential greedy strategy of the first algorithm has been used earlier for the k-VCSS problem by [9], and the parallel strategy of the second algorithm has been used earlier for the k-VCSS problem by [26]. Both algorithms rely on the Padded-outconnectivity algorithm; in general, the set of augmenting edges computed by the Padded-outconnectivity algorithm is *not* added to the current digraph; instead, we compute the cost of this edge set, and if it satisfies other criteria, then we add this edge set to the current digraph.

Proposition 35. *There is an $O(\log n)$-approximation algorithm for increasing (S, T) connectivity from ℓ to $\ell + 1$.*

□

The main result of this section, Theorem 4, follows from the above result and Proposition 27.

5.7 Approximation Algorithm 1

Our first algorithm decreases the number of cores by one in each iteration. Consider any iteration: For each core C, we apply the Padded-outconnectivity algorithm to compute a set of edges $F(C)$ that covers the halo family of C; but, at this point, we do not add any edges to the current digraph. We then choose a core C^* such that $c(F(C^*))$ is minimum, that is, $c(F(C^*)) = \min\{c(F(C)) : C \text{ is a core}\}$, and we add $F(C^*)$ to the current digraph. A key lemma below shows that the number of cores decreases by one in the resulting digraph. We repeat these iterations until no core is left in the current digraph; if there are no cores, then observe that the (S, T) connectivity of the digraph has increased by one.

The next result is a key result for the overall analysis, and several of the recent approximation algorithms for the k-VCSS problem rely on similar results ([24, 9, 26]). A new complication arises for our result, since the cores need not be disjoint, whereas the cores are disjoint in the previous literature. In general, when we add some augmenting edges, we cover some of the old cores, but the augmented digraph may have several new cores that are intersecting, e.g., there may exist $j \geq 2$ new cores that intersect each other but whose union contains less that j old cores; see Figure 7. Fortunately, the augmenting edge set found by the Padded-outconnectivity algorithm avoids such messy possibilities, as can be seen from the next result and its proof.

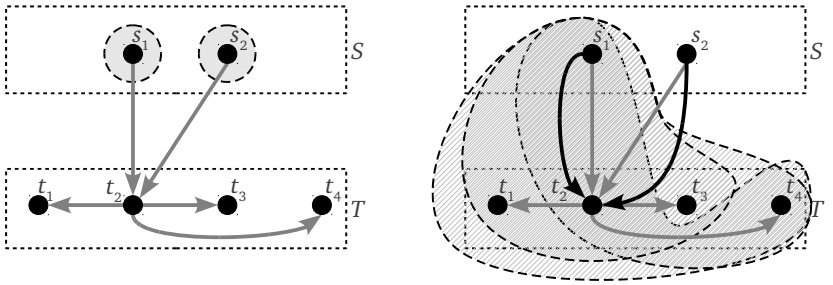


Figure 7: The figure shows two instances of the problem of increasing (S, T)-connectivity by one, where the initial digraph is 1-(S, T)-connected. The grey lines denote the edges of the initial digraph. The cores in the left instance are $\{s_1\}$ and $\{s_2\}$. In the right instance, $\{s_1\}$ and $\{s_2\}$ 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 $\{s_2, t_2, t_3, t_4\}$.

Lemma 36. *If we apply the Padded-outconnectivity algorithm to a core C and add the computed edge set $F(C)$ to the current digraph, then the number of cores decreases by at least one. In other words: Suppose that the digraph $(V, E_0 \cup E')$ has one or more cores. Let C be a core and let $F(C)$ be the set of augmenting edges found by the Padded-outconnectivity algorithm applied to C. Then the number of cores in $(V, E_0 \cup E' \cup F(C))$ is less than the number of cores in $(V, E_0 \cup E')$.*

Proof. We refer to the cores in the "old digraph" $(V, E_0 \cup E')$ as the *old cores*, and the cores in the "new digraph" $(V, E_0 \cup E' \cup F(C))$ as the *new cores*.

The proof follows from two key facts: (1) every one of the deficient sets in $\mathrm{Halo}(C)$ is covered by the set of augmenting edges $F(C)$; (2) every one of the old cores other than C is preserved, that is, except for C, all of the old cores are new cores. Fact (1) holds by definition; we will prove fact (2) below.

Consider any new core D'. Then D' is a deficient set in the old digraph $(V, E_0 \cup E')$, so it must contain one or more old cores. Suppose D' contains the old core C and no other old cores. Then $D' \in \mathrm{Halo}(C)$. By fact (1), D' must be covered by $F(C)$, thus D' cannot be a deficient set of the new digraph. In other words, there exist no new cores that contain C and no other old cores. Thus every new core must contain an old core $D \neq C$. However, by fact (2), every old core $D \neq C$ is a new core. We cannot have any other new cores, since one new core cannot properly contain another new core. Thus the lemma follows from facts (1) and (2).

Now, consider fact (2) and its proof. When the Padded-outconnectivity algorithm is applied to any core C, then it finds an (inclusionwise) minimal set of augmenting edges $F(C)$ that covers $\mathrm{Halo}(C)$; the minimality holds because the algorithm finds a set of augmenting edges of minimum cost. It then follows from Lemma 34 that for any core $D \neq C$, none of the deficient sets in $\mathrm{Halo}(D)$ is covered by any edge in $F(C)$. Thus every old core $D \neq C$ stays as a deficient set of the new digraph, and moreover, it must be a new core (it cannot properly contain some other deficient set of the new digraph, since it is a minimal deficient set of the old digraph).

□

5.8 Cost analysis by decomposing an optimal fractional solution

For the rest of this section, we revise our definitions of opt and E^*, for the sake of notational convenience. We use opt to denote the optimal value of $\mathrm{LP}^{inc}(\ell)$, which is the LP relaxation for increasing (S,T) connectivity from ℓ to $\ell+1$, and we use E^* to denote the support of some fixed optimal solution of this LP (thus, letting $\boldsymbol{x}$ denote an optimal solution of $\mathrm{LP}^{inc}(\ell)$, we have $E^* = \{e \in E : \boldsymbol{x}_e > 0\}$).

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 (inclusion-wise) minimal subset of E^* such that $\mathrm{Halo}(C_i)$ is covered by $\boldsymbol{x}$ restricted to $E^*(C_i)$.

Lemma 37 (Decomposition Lemma)**.** *$E^*(C_i)$ and $E^*(C_j)$ are disjoint for all $1 \leq i \neq j \leq t$. Furthermore, $\sum_{i=1}^{t} \sum_{e \in E^*(C_i)} c(e)\boldsymbol{x}(e) \leq \mathrm{opt}$.*

Proof. We prove the first statement by a contradiction argument. Suppose that the statement does not hold. Then there exist i, j with $1 \leq i < j \leq t$, such that $E^*(C_i) \cap E^*(C_j)$ contains an augmenting edge e. Then by the minimality of $E^*(C_i)$ and $E^*(C_j)$, e must cover some deficient set $U \in \mathrm{Halo}(C_i)$ as well as some deficient set $W \in \mathrm{Halo}(C_j)$. This contradicts Lemma 30 (the disjointness property). Hence, $E^*(C_i)$ and $E^*(C_j)$ are disjoint for all $i \neq j$.

The second statement is an immediate consequence of the first statement. □

Lemma 38. *The total cost incurred by the algorithm is $O(\log n)\mathrm{opt}$.*

Proof. Let t_0 denote the number of cores at the start of the algorithm; we have $t_0 \leq |S| \cdot |T| \leq n^2$, by Corollary 32. Each iteration decreases the number of cores by one, by Lemma 36. Moreover, we claim that the cost of the set of augmenting edges added by an iteration is $\leq \mathrm{opt}/t'$, where t' denotes the number of cores at the start of the iteration. This is proved below. Hence, the total cost of the edges added by the algorithm is $\leq \mathrm{opt}\ \left(\frac{1}{t_0} + \frac{1}{t_0-1} + \ldots + 1\right) = O(\log t_0)\ \mathrm{opt} = O(\log n)\ \mathrm{opt}$.

Claim 39. *Let t be the number of cores.*

(1) *Let $C_1, \ldots, C_t$ be all of the cores, and let $F(C_i)$ be an edge set of minimum cost that covers $\mathrm{Halo}(C_i)$, $\forall i = 1, \ldots, t$. Then $\sum_{i=1}^{t} c(F(C_i)) \leq \mathrm{opt}$.*

(2) *Let C^* be a core such that the edge set $F(C^*)$ has minimum cost, i.e., $c(F(C^*)) \leq c(F(C)), \forall$ cores C. Then $c(F(C^*)) \leq \mathrm{opt}/t$.*

Proof. Consider any core C. Recall that $F(C)$ denotes the set of augmenting edges found by the Padded-outconnectivity algorithm, and $E^*(C)$ denotes an (inclusion-wise) minimal subset of E^* such that $\mathrm{Halo}(C)$ is covered by $\boldsymbol{x}$ restricted to $E^*(C)$. Then we have $c(F(C)) \leq \sum_{e \in E^*(C)} c(e)\boldsymbol{x}(e)$. This follows from Frank's results on the LP relaxation for the min-cost k-(r,T) connected digraph problem. Frank proves that the LP relaxation is integral, see Theorem 12, and also see Theorems 4.4 and 5.9 of [14].

We apply this to all the cores $C_1, C_2, \ldots, C_t$. Thus, we have $c(F(C_i)) \leq \sum_{e \in E^*(C_i)} c(e)\boldsymbol{x}(e)$, for $i = 1, 2, \ldots, t$. Moreover, we have $\sum_{i=1}^{t} \sum_{e \in E^*(C_i)} c(e)\boldsymbol{x}(e) \leq \mathrm{opt}$, by Lemma 37. Hence,

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

Finally, observe that

$$c(F(C^*)) = \min_{i=1}^{t}\{c(F(C_i))\} \leq \min_{i=1}^{t}\left\{\sum_{e\in E^*(C_i)} c(e)\boldsymbol{x}(e)\right\} \leq \frac{\sum_{i=1}^{t}\sum_{e\in E^*(C_i)} c(e)\boldsymbol{x}(e)}{t} \leq \frac{\mathrm{opt}}{t}.$$

This proves both **(1)** and **(2)**, and the claim follows. □

This completes the proof of the lemma. □

Approximation Algorithm 1 together with its analysis gives a proof of Proposition 35.

Remark 40. Approximation Algorithm 1 can be implemented to run in time $O(n^6m + n^5m \cdot f(m,n))$, where $f(m,n)$ denotes the time for computing a maximum s,t flow.

5.9 Approximation Algorithm 2

The second approximation algorithm executes $O(\log n)$ rounds, where each round adds a set of augmenting edges with the hypothetical goal of decreasing the number of cores by a factor of two. At the start of each round, we compute the set $\mathcal{C}$ of all cores for the current digraph; then, for each core $C \in \mathcal{C}$, we compute the set of edges $F(C)$ that covers $\mathrm{Halo}(C)$, via the Padded-outconnectivity algorithm; then, we add all these edge sets to the current digraph, that is, we add the edge set $\bigcup\{F(C) \mid C \in \mathcal{C}\}$; this completes one round. We repeatedly apply such rounds until there is no core left.

Lemma 41. *No deficient set contains two cores whose bodies are intersecting.*

Proof. By Lemma 30 (the disjointness property), any two distinct cores C and D whose bodies are intersecting must have disjoint shadows. Hence, $C \supseteq T - D$, and $D \supseteq T - C$, thus $C \cup D$ contains T. Thus, any set of vertices containing $C \cup D$ cannot be a deficient set, because it contain T. □

Lemma 42. *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 at 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 every deficient set that is contained in some halo family. Thus, the current digraph has no deficient set that contains exactly one old core. In other words, any new core contains at least two old cores. By Lemma 41, a new core cannot contain two old cores whose bodies are intersecting because each new core is a deficient set in the old digraph. Hence, ν' body-disjoint new cores must contain at least $2\nu'$ body-disjoint old cores. Thus, $2\nu' \leq \nu$ which proves the lemma. □

Lemma 43. *The algorithm terminates within $O(\log n)$ rounds, and it runs in polynomial time. Moreover, the total cost incurred by the algorithm is at most $O(\log n)\mathrm{opt}$.*

Proof. The maximum number of body-disjoint cores is $O(|S|) = O(n)$, and the maximum number of body-disjoint cores decreases by half in each round, hence the number of rounds is $O(\log n)$.

The cost of the edges added in each round is at most opt. To see this, let $C_1, C_2, \ldots, C_t$ be all of the cores. Recall that, for $i = 1, 2, \ldots, t$, $F(C_i)$ is a set of edges of minimum cost that covers $\mathrm{Halo}(C_i)$, hence, by Claim 39, we have $\sum_{i=1}^{t} c(F(C_i)) \leq \mathrm{opt}$. Thus the total cost incurred in $O(\log n)$ rounds is $O(\log n)\mathrm{opt}$. □

Approximation Algorithm 2 together with its analysis gives another proof of Proposition 35.

Remark 44. Approximation Algorithm 2 can be implemented to run in time $O((n^4m + n^3m \cdot f(m,n)) \cdot \log n)$, where $f(m,n)$ denotes the time for computing a maximum s,t flow.

6 An approximation algorithm for relaxed 1-(S,T) connectivity

In this section, we present an approximation algorithm for the relaxed (S,T) connectivity problem. The problem reduces to the special case where there is no T,S dipath. Hence, we focus on this special case. Our approximation algorithm and its analysis are based on a key structural result that decomposes any feasible solution into a set of junction trees that are disjoint on the vertices of T. Our algorithm achieves an approximation guarantee of $\alpha(n)+1$, where $\alpha(n)$ denotes the best available approximation guarantee for the directed Steiner tree problem; see Theorem 9. Our approximation guarantee is tight up to an additive term of one, since Theorem 1 shows that the relaxed (S,T) connectivity problem is at least as hard as the directed Steiner tree problem. We state the main result of this section.

Theorem 5. *Consider the relaxed k-(S,T) connectivity problem with the connectivity parameter k equal to one. There exists a $(\alpha(n)+1)$-approximation algorithm, where $\alpha(n)$ denotes the (best available) approximation guarantee for the directed Steiner tree problem. In particular, there is an $O(\log^3 n)$-approximation algorithm that runs in quasi-polynomial time.*

There is a simple, linear-time reduction from the relaxed (S,T) connectivity problem to its special case where there is no T,S dipath. For each vertex $s \in S$, we add a new vertex s^+ and a new edge (s^+,s) to G_0 (the initial digraph); the vertex s and its other incident edges stay the same. Then we replace each vertex s in S by the associated vertex s^+, to get $S^{new} = \{s^+ : s \in S\}$. It is easily seen that a set of edges $\widehat{E} \subseteq E$ is a solution to the new instance iff it is a solution to the original instance. Observe that the new instance has no T, S^{new} dipath, because each of the vertices s^+ in S^{new} has indegree zero.

6.1 Relaxed (S,T) connectivity: An approximation algorithm for the case of no T,S dipath

In this section, we assume that there is no T,S dipath in G. We start with a summary of our algorithm, in order to motivate our key structural result on decomposing a feasible solution. Based on the structural result, we design and analyse an $(\alpha(n)+1)$-approximation algorithm for the problem.

We need the notion of junction trees. Let r be a vertex. Recall that an in-tree J^{in} rooted at r is a digraph that has a v,r dipath for every vertex $v \in V(J^{in})$, and similarly, we have the notion of an out-tree J^{out} rooted at r. A *junction tree J rooted at r* is the union of an in-tree J^{in} and an out-tree J^{out}, both rooted at the same vertex r. The in-tree and the out-tree may have common edges.

Our algorithm constructs an auxiliary digraph, then computes a rooted out-branching M of minimum-cost in it, and then maps M back to the original digraph G to get a solution to the relaxed (S,T) connectivity problem. The auxiliary digraph is constructed as follows. For each vertex $t \in T$, we compute an in-directed Steiner tree F_t rooted at t with terminal set S of approximately minimum cost. Then we remove all incident edges from S, contract S to a single vertex $\widehat{s}$, and add an edge $(\widehat{s},t)$ of cost $c(F_t)$ for each $t \in T$; finally, we remove all optional vertices $v \in V - (S \cup T)$ that are not reachable from T; this completes the construction of the auxiliary digraph. Then we compute a min-cost out-branching M with root $\widehat{s}$ for the auxiliary digraph; observe that all vertices are reachable from $\widehat{s}$, by construction. Finally, we replace each edge $(\widehat{s},t)$ of M by the corresponding directed Steiner tree F_t to get a solution digraph $\widehat{G}$. We analyse the cost of $\widehat{G}$ by

comparing it to the cost of an optimal solution $G^* = G_0 \cup E^*$. To handle this comparison, we decompose G^* into junction trees that are disjoint on T, thus disjoint with respect to the set of augmenting edges.

Figure 8 illustrates the working of our algorithm.

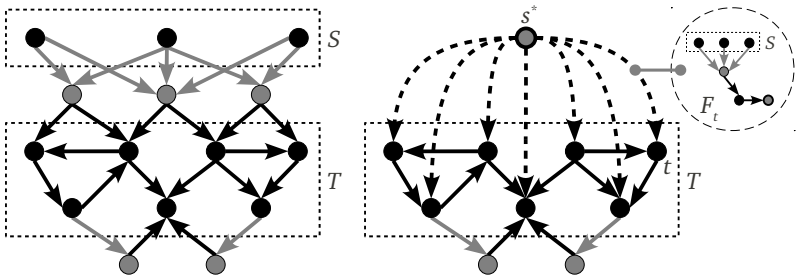


Figure 8: The figure illustrates the working of our approximation algorithm for the relaxed (S, T) connectivity problem on an example that has no T, S dipath. The left figure shows the original digraph. The right figure shows the auxiliary digraph. The set of vertices S is contracted into a single vertex $\widehat{s}$. 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 dash-lines denote auxiliary edges obtained by replacing an in-directed Steiner tree F_t rooted at a vertex $t \in T$ by an edge $(\widehat{s}, t)$ with cost $c(F_t)$ for each vertex $t \in T$.

6.1.1 Decomposing a feasible solution

We give a structural result that applies to any feasible solution of the relaxed (S, T) connectivity problem. We prove our approximation guarantee by applying this result to an optimal solution, hence, we consider an optimal solution $G^* = (V, E_0 \cup E^*)$ to the relaxed (S, T) connectivity problem. But, we remark that the results in this subsection (Lemmas 45 and 46) apply for any feasible solution.

We 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 an s, t dipath for all $s \in S$ and all $t \in V(J_i) \cap T$.
- 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, given an optimal set of augmenting edges E^*, we want to partition it into subsets $E_1^*, \ldots, E_\ell^*$ such that each subset E_i^* together with E_0 forms a junction tree J_i connecting S and $V(J_i) \cap T$.

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 G has no T, S dipath; moreover, any two maximal strongly-connected components have no common vertices; otherwise, the two would have been merged. We abuse the notation and continue using the same symbols for the contracted digraph. At this point, the contracted digraph G^* is acyclic. Hence, there exists a vertex $t^* \in T$ such that there exists no t, t^* dipath for any other vertex $t \in T$. We call such a vertex t^* a *top-vertex*.

The construction runs in several iterations on the contracted digraph. 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 the root of

the junction tree to be a top-vertex t_i of the current digraph. The out-tree of J_i consists of dipaths from t_i to all vertices of T reachable from t_i in the current digraph. Observe that every dipath (of the current digraph) from S to $V(J_i) \cap T$ must visit t_i before reaching any other vertex of $V(J_i) \cap T$, because t_i is a top-vertex; thus, $J_i - \{t_i\}$ has no S, T dipaths. Then, we remove the vertices of T that are assigned to J_i from the current digraph. We repeat this process until each vertex of T is assigned to some junction tree.

In more detail, we start with the contracted 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, and J_i consists of an in-tree J_i^{in} and an out-tree J_i^{out}, that is, $J_i = J_i^{in} \cup J_i^{out}$. Both J_i^{in} and J_i^{out} are subgraphs of G_{i-1}^*. The in-tree J_i^{in} is obtained by taking an in-directed Steiner tree of G_{i-1}^* rooted at t_i with terminal set S. The out-tree J_i^{out} is obtained by taking the union of t_i, t dipaths for all vertices $t \in T_{i-1}$ reachable from t_i in G_{i-1}^*. Once we have the junction tree J_i, we update the digraph G_{i-1}^* and the terminal set T_{i-1} by removing all vertices of T assigned to J_i. Thus, we have $G_i^* = G_{i-1}^* - V(J_i) \cap T$ and $T_i = T_{i-1} - V(J_i) \cap T$. We continue to the next iteration and repeat the process until all vertices of T are assigned to junction trees. The stopping condition is $T_\ell = \emptyset$, or equivalently, T is contained in $\bigcup_{i=1}^{\ell} J_i$.

At termination, we uncontract the strongly-connected components of G^*. Suppose that the root t_i of a junction tree J_i in the contracted digraph corresponds to a nontrivial strongly-connected component C_i; then, when we uncontract the digraph, we take the root t_i to be any vertex of T in C_i. It can be seen that this preserves the three properties of junction trees listed above. We remark that the in-tree and the out-tree may have common vertices of T in the uncontracted digraph, hence, we may have $c(J_i) < c(J_i^{in}) + c(J_i^{out})$.

Clearly, the junction trees $J_1, J_2, \ldots, J_\ell$ of the uncontracted digraph have no common vertices of T, and so they have no common augmenting edges. This implies that $\sum_{i=1}^{\ell} c(J_i) = c(E^*) = \mathrm{opt}$.

The next two results prove that these junction trees satisfy the required properties.

Lemma 45. *At the iteration i, $i = 1, 2, \ldots, \ell$, the digraph G_{i-1}^* is (S, T_{i-1}) connected.*

Proof. The proof hinges on a key property of a junction tree J_i: every node of $V(J_i) \cap T$ is reachable from the root t_i.

We 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 a feasible solution.

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 $t \in T_i$ such that G_i^* has no s, t dipath. By the induction hypothesis, G_{i-1}^* is (S, T_{i-1}) connected and so has an s, t dipath P. Then P must contain some vertex of $V(J_i) \cap T$, otherwise, P is also contained in $G_i^* = G_{i-1}^* - V(J_i) \cap T$. Then G_{i-1}^* has a dipath from t_i to t, because J_i has dipaths from t_i to each vertex of $V(J_i) \cap T$, and so the union of J_i and P contains a t_i, t dipath. By the construction of J_i, the vertex t must be included in J_i and thus must have been removed from G_i^*, a contradiction. Therefore, G_i^* is (S, T_i) connected. □

Lemma 46. *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 an s, t dipath for all $s \in S$ and all $t \in V(J_i) \cap T$. Moreover, if the original digraph G is acyclic on T, then J_i^{in} has exactly one vertex of T, namely, the root t_i.

(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 45. Consider any $i = 1, 2, \ldots, \ell$. 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 with terminal set S. Moreover, J_i^{out} has a t_i, t dipath, for every vertex $t \in V(J_i^{out}) \cap T$.

Now, focus on the contracted digraph obtained by contracting all maximal strongly-connected components of G^*. We claim that t_i is the unique vertex of T in J_i^{in}, that is, $V(J_i^{in}) \cap T = \{t_i\}$. Note that J_i^{in} is an in-directed Steiner tree in G^*_{i-1} rooted at t_i. Hence, if J_i^{in} contains some other vertex $t' \in T$, then it has a t', 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 dipath from $(T_{i-1} - \{t_i\})$ to t_i.

In general, the original (uncontracted) digraph G^* may have two or more vertices of T in J_i^{in}. In the special case where the digraph G is acyclic on T, observe that every strongly-connected component has at most one vertex of T. Hence, in this special case, the above property of the contracted digraph is preserved even after we uncontract the strongly-connected components, that is, t_i is the unique vertex of T in J_i^{in}.
(ii) The second property holds because we remove all vertices of $V(J_i) \cap T$ from the digraph G^*_{i-1} before proceeding to the next iteration, for $i = 1, 2, \ldots, \ell$. Moreover, no two junction trees have a common augmenting edge, because each vertex of T is in exactly one junction tree, and all augmenting edges have heads in T.
(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. □

6.1.2 An approximation algorithm

A formal description of our algorithm follows.

1. Construct an auxiliary digraph G^{aux} with vertex set $\{\widehat{s}\} \cup T \cup Q$, where $\widehat{s}$ is a new vertex that represents S, and Q denotes the set of all optional vertices $v \in V - (S \cup T)$ that are reachable from T in G; G^{aux} has all the edges of G that have both end-vertices in $T \cup Q$ with the same costs, and in addition G^{aux} has a so-called *auxiliary edge* $(\widehat{s}, t)$ for each vertex $t \in T$. Note that all vertices are reachable from $\widehat{s}$ in G^{aux}.

2. For each vertex $t \in T$, compute an in-directed Steiner tree F_t of G rooted at t with terminal set S of approximately minimum cost (see Theorem 9), and then define the cost of the auxiliary edge $(\widehat{s}, t)$ to be $c(F_t)$.

3. Compute a minimum cost out-branching M of G^{aux} rooted at $\widehat{s}$.

4. Informally speaking, we map M back to G to get a solution digraph $\widehat{G} = (V, E_0 \cup \widehat{E})$, by replacing each auxiliary edge $(\widehat{s}, t)$ in M by the corresponding in-directed Steiner tree F_t. Formally, we take $\widehat{E}$ to consist of (a) all edges of M that have both end-vertices in $T \cup Q$, and (b) all of the augmenting edges of each in-directed Steiner tree F_t such that the auxiliary edge $(\widehat{s}, t)$ is in M; thus, we have $\widehat{E} = (M - E_0 - \{(\widehat{s}, t) : t \in T\}) \cup \bigcup_{t:(\widehat{s},t) \in M} (F_t - E_0)$. Observe that $c(\widehat{E}) \leq c(M)$.

The next result gives the correctness and cost analysis for the algorithm.

Proposition 47. *The above algorithm finds a feasible solution of cost* $\leq (\alpha(n)+1)\mathrm{opt}$ *for the relaxed* (S,T) *connectivity problem.*

Proof. The correctness of our solution follows from the fact that M is an out-branching rooted at $\widehat{s}$. In more detail, consider any vertex $t \in T$. Observe that every $\widehat{s},t$ dipath in M is of the form $\widehat{s} \rightarrow t^* \rightarrow \ldots \rightarrow t$, where $(\widehat{s},t^*)$ is an auxiliary edge while the other edges belong to the digraph G. Since we replace $(\widehat{s},t^*)$ by the in-directed Steiner tree F_{t^*} in the final step, the resulting digraph $\widehat{G}$ must have an s,t^* dipath for every $s \in S$. Hence, we have an s,t dipath of the form $s \rightarrow \ldots \rightarrow t^* \rightarrow \ldots \rightarrow t$, for each vertex $s \in S$. Thus, the resulting digraph $\widehat{G}$ is (S,T) connected.

Now, consider the cost analysis. We abbreviate $\alpha(n)$ to α within this proof. We have $c(\widehat{E}) \leq c(M)$. Our key claim is that $c(M) \leq (\alpha+1)\mathrm{opt}$. To prove this, we start with the digraph $G^* = (V, E_0 \cup E^*)$ of an optimal solution E^* and construct a spanning subgraph M^* of G^{aux} such that M^* contains an out-branching of G^{aux} rooted at $\widehat{s}$ and $c(M^*) \leq (\alpha+1)\mathrm{opt}$; clearly, this will prove the claim. We apply Lemma 46 to G^* to obtain junction trees $J_1,\ldots,J_\ell$ that satisfy properties (i)–(iii) of the lemma. Recall that t_i denotes the root of the in-directed Steiner tree J_i^{in} of J_i, for $i=1,\ldots,\ell$. For each of the junction trees J_i, where $i=1,\ldots,\ell$, we add the auxiliary edge $(\widehat{s},t_i)$ to M^*; observe that $c(\widehat{s},t_i) \leq \alpha\, c(J_i^{in})$ because J_i^{in} is an in-directed Steiner tree of G rooted at t_i with terminal set S and $c(\widehat{s},t_i) \leq \alpha\, c^*_{t_i}$, where $c^*_{t_i}$ denotes the minimum cost of an in-directed Steiner tree of G rooted at t_i with the same terminal set as J_i^{in}.

Next, for $i=1,\ldots,\ell$, we add to M^* all the edges of J_i^{out}. Observe that J_i^{out} is a subgraph of G^{aux} because each edge in J_i^{out} is reachable from t_i, hence, both end-vertices of the edge are in $T \cup Q$. Since the junction trees $J_1,\ldots,J_\ell$ satisfy properties (i)–(iii) (of Lemma 46), it follows that M^* contains T, and moreover, M^* has an $\widehat{s},t$ dipath for each vertex $t \in T$.

Finally, we add to M^* the edges of a minimum-cost dipath from T to v, for each optional vertex $v \in Q$; each of these edges has cost zero since the head is in Q. This completes the construction of M^*. Clearly, M^* contains an out-branching of G^{aux} rooted at $\widehat{s}$. We have

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

This implies that our algorithm achieves an approximation guarantee of $(\alpha(n)+1)$ as required. We remark that the additive term of $+1$ arises because $c(J_i)$ may be strictly less than $c(J_i^{in}) + c(J_i^{out})$, since J_i^{in} and J_i^{out} may have common vertices of T. □

6.2 The relaxed (S,T) connectivity problem on a digraph that is acyclic on T

In this section, we focus on the special case of the relaxed (S,T) connectivity problem where the digraph G is acyclic on T. First, we show that this problem is at least as hard for approximation as SCP. Then, we refine the algorithm of Section 6.1 to improve the approximation guarantee to $O(\log|S|)$ when the digraph is acyclic on T.

For the hardness result, consider an instance I_{SC} of SCP with ground-set $U = \{u_1,\ldots,u_p\}$ and subsets $S_1,\ldots,S_q \subseteq U$. We can represent I_{SC} by a bipartite graph B whose vertex partition consists of U and $W = \{S_1,\ldots,S_q\}$; B has an edge between $u_i \in U$ and $S_j \in W$ iff $u_i \in S_j$ in the instance I_{SC}. To obtain an instance of relaxed (S,T) connectivity, we orient the edges of B from U to W, and then we add a new vertex $\widehat{t}$ and the edges $(S_j,\widehat{t})$ with cost $c(S_j)$ for each subset S_j of the instance I_{SC}; we give a cost of zero to all other edges, and we fix $T = \{\widehat{t}\}$, $S = U$; note that each edge of positive cost has its head in T. This completes the construction. It can be seen that a feasible solution $\widehat{E}$ for the instance of relaxed

(S,T) connectivity corresponds to a feasible solution of the instance I_{SC} of the same cost, by choosing a subset S_j of I_{SC} iff an edge $(S_j, \widehat{t})$ is in $\widehat{E}$.

Consider the refined approximation algorithm for relaxed (S,T) connectivity. For any vertex $\widehat{t} \in T$, when we compute an in-directed Steiner tree with root $\widehat{t}$ and terminal set S, then we only consider solution subgraphs such that all augmenting edges have heads at $\widehat{t}$; in other words, we ignore solution subgraphs that contain $s, \widehat{t}$ dipaths that use ≥ 2 augmenting edges for some $s \in S$. To see the correctness of this construction, consider an optimal solution $G^* = G_0 + E^*$ and any top vertex $\widehat{t}$ in the "decomposition" given by Lemma 46. The in-directed Steiner tree $J^{in}_{\widehat{t}}$ (of the decomposition) with root $\widehat{t}$ and terminal set S contains no other vertices of T, by property (i) of Lemma 46, hence, $J^{in}_{\widehat{t}}$ contains no augmenting edge with head in $T - \{\widehat{t}\}$. There may exist vertices $\widehat{t} \in T$ such that there exist no in-directed Steiner trees rooted at $\widehat{t}$ satisfying the above conditions; then, we give an infinite cost to the corresponding auxiliary edges $(\widehat{s}, \widehat{t})$ in the auxiliary digraph constructed by the algorithm.

We can compute an approximately min-cost directed Steiner tree of this special form by solving the following instance I_{SC} of SCP. We take the ground-set in the instance I_{SC} to be the set S (in the instance of relaxed (S,T) connectivity); moreover, for each edge e_j with head at $\widehat{t}$ (in the instance of relaxed (S,T) connectivity), we have a subset $S_j \subseteq S$ in the instance I_{SC}, where $S_j = \{s \in S : G_0 + e_j \text{ has an } s, \widehat{t} \text{ dipath}\}$.

A greedy algorithm for SCP gives an approximation guarantee of $O(\log |S|)$ for each of these instances of SCP (one for each vertex in T). Hence, by Proposition 47, the algorithm for relaxed (S,T) connectivity achieves an approximation guarantee of $O(\log |S|)$. The next result follows.

Theorem 6. *Consider the special case of the relaxed (S,T) connectivity problem (thus $k = 1$) where the given digraph G is acyclic on T. This problem is at least as hard as the set covering problem. Moreover, there exists an $O(\log |S|)$-approximation algorithm for this problem.*

7 Conclusions

We introduced the model of k-(S,T) connectivity to capture some NP-hard problems in network design, and we designed approximation algorithms for some of the key problems, [25, 4]. Our algorithms are simple and versatile. One of our algorithmic contributions is to show that the halo-set method gives surprisingly good results, even when feasible solutions have less structure than those of the k-VCSS problem where the method was originally applied.

Acknowledgments. We thank our colleagues for useful comments. In particular, we thank Chaitanya Swamy and Jochen Könemann who were readers for the second author's thesis which contains many of the results of this paper.

References

[1] S.Arora and C.Lund, Hardness of Approximations. In *Approximation algorithms for NP-hard problems*, Ed. D.S.Hochbaum, PWS publishing co., Boston, 1996.

[2] J.Bang-Jensen and G.Z.Gutin, *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2008.

[3] M.Charikar, C.Chekuri, T.-Y.Cheung, Z.Dai, A.Goel, S.Guha, and M.Li, Approximation algorithms for directed steiner problems. In *SODA*, pages 192–200, 1998.

[4] J.Cheriyan, Combinatorial optimization on k-connected graphs. In Proc. *ICRTGC – ICM satellite conference*, Aug. 2010.

[5] J.Cheriyan, and A.Vetta, Approximation algorithms for network design with metric costs. *SIAM J. Disc. Math.*, 21(3):612–636, 2007.

[6] J.Chuzhoy and S.Khanna, An $O(k^3 \log n)$-approximation algorithm for vertex connectivity survivable network design. In *FOCS*, pages 437–441. IEEE, 2009.

[7] R.Diestel, *Graph Theory*. Springer, Berlin, 2006.

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

[9] J.Fakcharoenphol and B.Laekhanukit, An $O(\log^2 k)$-approximation algorithm for the k-vertex connected spanning subgraph problem. In R.E. Ladner and C. Dwork, editors, *STOC*, pages 153–158. ACM, 2008.

[10] U.Feige, A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[11] L.Fleischer, K.Jain, and D.P.Williamson, An iterative rounding 2-approximation algorithm for the element connectivity problem. *Journal of Computer and System Sciences*, 72 (5), pp. 838–867, 2006. Preliminary version in *Proc. 42nd IEEE FOCS*, pp.339-347, 2001.

[12] A.Frank, Connectivity augmentation problems in network design, in *Mathematical Programming: State of the Art 1994*, (Eds.J.R.Birge and K.G.Murty), pp.34–63, The University of Michigan, Ann Arbor, MI, 1994.

[13] A.Frank, Increasing the rooted-connectivity of a digraph by one. *Mathematical Programming*, 84(3):565–576, 1999. Connectivity augmentation of networks: structures and algorithms (Budapest, 1994).

[14] A.Frank, Rooted k-connections in digraphs. *Discrete Applied Mathematics*, 157(6):1242–1254, 2009.

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

[16] A.Frank and E.Tardos, An application of submodular flows, *Linear Algebra and its Applications*, 114/115:329–348, 1989.329-348.

[17] G.Frederickson and J.JáJá, Approximation algorithms for several graph augmentation problems, *SIAM J. Computing*, 10(2):270–283, 1981.

[18] M.X. Goemans, A.V. Goldberg, S.A. Plotkin, D.B. Shmoys, E. Tardos, and D.P. Williamson, Improved approximation algorithms for network design problems. In *SODA*, pages 223–232, 1994.

[19] E. Halperin and R. Krauthgamer, Polylogarithmic inapproximability. In *STOC*, pages 585–594, 2003.

[20] C.S. Helvig, G. Robins, and A. Zelikovsky, An improved approximation scheme for the group steiner problem. *Networks*, 37(1):8–20, 2001.

[21] S.Khuller, Approximation algorithms for finding highly connected subgraphs. In *Approximation algorithms for NP-hard problems*, Ed. D.S.Hochbaum, PWS publishing co., Boston, 1996.

[22] S.Khuller and B.Raghavachari, Improved approximation algorithms for uniform connectivity problems. *Journal of Algorithms* 21, 434–450, 1996.

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

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

[25] B.Laekhanukit, Approximation algorithms for (S,T)-connectivity problems. M.Math. Thesis, University of Waterloo, 2010.

[26] Z.Nutov, An almost $O(\log k)$-approximation for k-connected subgraphs. In Claire Mathieu, editor, *SODA*, pages 912–921. SIAM, 2009.

[27] Z.Nutov, Approximating minimum cost connectivity problems via uncrossable bifamilies and spider-cover decompositions. In *FOCS*, pages 417–426. IEEE, 2009.

[28] R. Ravi and D.P. Williamson, An approximation algorithm for minimum-cost vertex-connectivity problems. *Algorithmica*, 18(1):21–43, 1997. Preliminary version in Proc. 6th ACM-SIAM SODA, 332–341, 1995. Erratum. *Algorithmica*, 34:98–107, 2002.

[29] A. Schrijver, Min-max relations for directed graphs. In: Bonn Workshop on Combinatorial Optimization (Bonn, 1980), Annals of Discrete Mathematics, 16:261–280, North-Holland, Amsterdam, 1982.

[30] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics, Vol. 24. Springer, Berlin, 2003.

[31] V.V.Vazirani, *Approximation Algorithms*. Springer, Berlin, 2001.

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

[33] D.P.Williamson and D.B.Shmoys, *The Design of Approximation Algorithms*. Cambridge University Press, 2010.