Algorithms for Single-Source Vertex Connectivity

Julia Chuzhoy
Toyota Technological Institute
Chicago, IL 60637
cjulia@tti-c.org

Sanjeev Khanna*
University of Pennsylvania
Philadelphia PA 19103
sanjeev@cis.upenn.edu

Abstract

In the Survivable Network Design Problem (SNDP) the goal is to find a minimum cost subset of edges that satisfies a given set of pairwise connectivity requirements among the vertices. This general network design framework has been studied extensively and is tied to the development of major algorithmic techniques. For the edge-connectivity version of the problem, a 2-approximation algorithm is known for arbitrary pairwise connectivity requirements. However, no non-trivial algorithms are known for its vertex connectivity counterpart. In fact, even highly restricted special cases of the vertex connectivity version remain poorly understood.

We study the single-source k-vertex connectivity version of SNDP. We are given a graph $G(V, E)$ with a subset T of terminals and a source vertex s, and the goal is to find a minimum cost subset of edges ensuring that every terminal is k-vertex connected to s. Our main result is an $O(k \log n)$-approximation algorithm for this problem; this improves upon the recent $2^{O(k^2)} \log^4 n$-approximation. Our algorithm is based on an intuitive rerouting scheme. The analysis relies on a structural result that may be of independent interest: we show that any solution can be decomposed into a disjoint collection of multiple-legged spiders, which are then used to re-route flow from terminals to the source via other terminals.

We also obtain the first non-trivial approximation algorithm for the vertex-cost version of the same problem, achieving an $O(k^7 \log^2 n)$-approximation.

1. Introduction

In the Survivable Network Design problem (SNDP) we are given a graph $G = (V, E)$ with costs on edges and integral connectivity requirements $r_{u,v} \geq 0$ for all $u, v \in V$. The goal is to find a minimum-cost subset $E' \subseteq E$ of edges, such that every pair u, v of vertices has at least $r_{u,v}$ disjoint paths connecting them in the graph induced by E'. There are two basic versions of SNDP: in the edge connectivity version (EC-SNDP), the $r_{u,v}$ paths connecting u and v are required to be edge disjoint, while in the vertex connectivity version (VC-SNDP) they are required to be vertex disjoint. We denote by n the number of vertices in the graph, and by k the largest connectivity requirement, $k = \max_{u,v} \{r_{u,v}\}$.

This general framework is very versatile, and several classical optimization problems, such as minimum spanning tree and minimum Steiner tree are captured as special cases of SNDP. Consequently, the problem has received considerable attention, and in fact the study of SNDP is linked to the development of several fundamental paradigms in algorithm design. Agrawal, Klein and Ravi [1] showed a 2-approximation algorithm for the restricted version of SNDP where $r_{u,v} \in \{0, 1\}$ for all u, v (notice that for this case VC-SNDP and EC-SNDP are equivalent). This result is among the first applications of the *primal-dual* paradigm to approximation algorithms, and this approach was later successfully used to design algorithms for higher connectivity values for EC-SNDP [18, 16], eventually leading to an $O(\log k)$-approximation [17]. The best current approximation ratio of 2 is achieved by an *iterative* LP-rounding algorithm due to Jain [19]. Both the primal-dual schema and the iterative rounding technique have been extensively used in approximation algorithm design.

It is not hard to see that EC-SNDP can be cast as a special case of VC-SNDP. Despite the fact that the edge version has been extensively studied and is well understood today, little is known about VC-SNDP. To the best of our knowledge, no approximation algorithm is known for this problem, outside for the trivial algorithm, that finds, for each $u, v \in V$, the cheapest collection of $r_{u,v}$ vertex disjoint paths and outputs the union of these paths. On the hardness side, Kortsarz *et. al.* [22] showed that, in sharp contrast to the edge connectivity version, VC-SNDP is hard to approximate up to $2^{\log^{1-\epsilon} n}$ factor for any $\epsilon > 0$, when k is polynomially large

*Supported in part by a Guggenheim Fellowship, an IBM Faculty Award, and by NSF Award CCF-0635084.

in n. This has been recently improved by Chakraborty *et. al.* [7] to a k^{ϵ}-hardness of approximation for all $k > k_0$, where ϵ, k_0 are fixed constants. The restricted special case where all connectivity requirements $r_{u,v} \in \{0, 1, 2\}$ was shown to have a 2-approximation algorithm by Fleischer *et. al.* [13]. Their algorithm is based on an application of the iterative rounding technique of Jain [19] to the set-pair LP-relaxation of Frank and Jordan [14]. Our current lack of understanding of VC-SNDP is perhaps best highlighted by the following state of affairs. When each connectivity requirement $r_{u,v} \in \{0, 3\}$, the best known approximation factor is polynomially large while nothing more than APX-hardness is known on the hardness side.

In this paper we focus on the *single-source version* of VC-SNDP, where we are given a set $T \subseteq V$ of terminals and a special vertex $s \in V \setminus T$ called the *source*. The only non-zero connectivity requirements are between the terminals and the source, i.e., $r_{s,t} > 0$ for all $t \in T$ and all other requirements are 0. When $r_{s,t} = k$ for all $t \in T$, we refer the problem as the *single-source k-vertex connectivity* problem.

Even for this restricted case of VC-SNDP, little has been known until recently. A trivial $O(n)$ approximation can be achieved by connecting each terminal $t \in T$ to the source by cheapest collection of k vertex disjoint paths, that can be found via min-cost flow computations. Until recently, even for constant values of $k \geq 3$, only an $O(n)$-approximation was known with no super-constant hardness bounds. On the other hand, the current best inapproximability factor of $\Omega(\log n)$, due to Kortsarz *et. al.* [22], only holds when k is polynomially large in n. Only recently, Chakraborty *et. al.* [7] obtained the first non-trivial approximation ratios for $k \geq 3$; they give an $2^{O(k^2)} \log^4 n$-approximation algorithm for single-source k-vertex connectivity. Concurrently and independently of this work, Chekuri and Korula [8], building on the ideas in [7], have recently given an $O(k^{O(k)} \log n)$-approximation algorithm for single-source k-vertex connectivity. Their approach is based on analyzing the dual of the natural LP relaxation for the problem. They also obtain a similar approximation guarantee for the more general single-source rent-or-buy network design problem.

We also study the vertex-cost variation of VC-SNDP where the costs are on vertices instead of edges. Specifically, the goal is to find a minimum-cost subset $V' \subseteq V$ of vertices, such that in the graph induced by V', for every pair $u, v \in V$ of vertices there are $r_{u,v}$ vertex disjoint paths connecting u to v. We focus again on the single-source version. When $k = 1$ the problem becomes equivalent to the node-weighted Steiner tree problem, for which $O(\log n)$-approximation is known [21]. On the other hand, even this special case can be shown to be $\Omega(\log n)$-hard by a reduction from Set Cover problem [25, 12]. It is interesting that even for $k = 1$ the edge and the vertex weighted versions exhibit such different behavior.

Related Work We note that for some special cases of VC-SNDP better algorithms are known. The k-vertex connected spanning subgraph problem, a special case of VC-SNDP where for all $u, v \in V$ $r_{u,v} = k$, has been studied extensively. Cheriyan *et al.* [5, 6] gave an $O(\log k)$-approximation algorithm for this case when $k \leq \sqrt{n/6}$, and an $O(\sqrt{n/\epsilon})$-approximation algorithm for $k \leq (1 - \epsilon)n$. For large k, Kortsarz and Nutov [24] improved the preceding bound to an $O(\ln k \cdot \min\{\sqrt{k}, \frac{n}{n-k} \ln k\})$-approximation. Fakcharoenphol and Laekhanukit [11] improved it to an $O(\log n \log k)$-approximation, and further obtained an $O(\log^2 k)$-approximation for $k < n/2$. The iterative rounding technique has been used to give a 2-approximation for arbitrary r_{ij} values for a variant of VC-SNDP known as the *element connectivity* problem [13, 6]. Frank and Tardos [15] gave a polynomial time algorithm for finding a minimum cost k-outconnected subdigraph of a directed graph. This result has been used to obtain a 2-approximation algorithm for single-source k-vertex connectivity when T contains all vertices in G except the source, and the costs form a metric [20]. Better approximation ratios have also been obtained for variants of vertex-connectivity problems assuming uniform cost or metric cost on the edges (see, for example, [20, 23, 4]).

Our Results Our main result is stated below.

Theorem 1 *There is a randomized polynomial time $O(k \log n)$-approximation algorithm for the edge cost version of single-source k-vertex connectivity.*

This result improves upon the $2^{O(k^2)} \log^4 n$-approximation by Chakraborty *et. al.* [7]. Our algorithm and its analysis relies on a new result about the structure of solutions for single-source k-vertex connectivity.

For the vertex cost version we obtain a similar result, albeit with somewhat weaker ratios.

Theorem 2 *There is a randomized polynomial time $O(k^7 \log^2 n)$-approximation algorithm for the vertex cost version of single-source k-vertex connectivity.*

To the best of our knowledge, no non-trivial approximation algorithm was previously known for the vertex cost version.

The subset k-connectivity problem is a special case of VC-SNDP, where we are given a subset $T \subseteq V$ of terminals, and connectivity requirements are $r_{t,t'} = k$ for all $t, t' \in T$, with all other requirements being 0. Any α-approximation algorithm for the single source k-vertex connectivity problem can be converted into a $k\alpha$-approximation algorithm for subset k-connectivity problem, in both edge and vertex cost scenarios (see e.g. Theorem 7 in [7]). It is also easy to see

that an α-approximation algorithm for the single-source k-vertex connectivity implies a $k\alpha$-approximation for single-source VC-SNDP, where $r_{t,s} \in \{0, 1, ..., k\}$ can be arbitrary, in both edge cost and vertex cost scenarios. We thus obtain the following results:

Theorem 3 *In the edge cost model, there is a randomized polynomial time $O(k^2 \log n)$-approximation algorithm for both single-source VC-SNDP and the subset k-connectivity problem. In the vertex cost model, there is a randomized polynomial time $O(k^8 \log^2 n)$-approximation algorithm for both single-source VC-SNDP and the subset k-connectivity problem.*

Techniques It is easy to obtain an $O(|T|)$-approximation for both edge and vertex cost versions of single-source k-vertex connectivity: for each terminal $t \in T$, find minimum cost k vertex disjoint paths connecting t to s and output the union of all such paths for all $t \in T$. The worst case scenario for such an algorithm is when the same edge is used by many terminals when connecting to the source in the optimal solution. Since the algorithm connects each one of the terminals to the source separately, without considering possible sharing of edges by the terminals, we may end up paying $|T|$ times the cost of the optimal solution. However, if edges are heavily shared by the terminals in the optimal solution (when connecting to the source), the solution induces high connectivity among the terminals. This observation motivates our algorithm. We compute, for each terminal $t \in T$, the cheapest collection of k vertex disjoint paths connecting t to vertices in $(T \cup \{s\}) \setminus \{t\}$. Let $E(t)$ be the set of edges participating in these paths. We identify a large subset T' of terminals, such that every $t \in T'$ is k-vertex connected to $(T \cup \{s\}) \setminus T'$ by edges in $E(t)$, and solve the problem recursively on $T \setminus T'$.

The heart of the analysis lies in proving that total cost of edges in sets $E(t)$, $t \in T$ is bounded by $O(k \cdot \text{OPT})$. The main ingredient of our proof is a *prefix decomposition theorem*, which may be of independent interest. Consider an optimal solution OPT, and for each terminal $t \in T$, let $B(t)$ be a k-tuple of vertex disjoint paths connecting t to s. A path p is said to be a prefix of $f \in B(t)$ iff p is a sub-path of f containing t. We show that we can define, for each $t \in T$ and $f \in B(t)$ a prefix $p(f)$ of f, such that the resulting set of prefixes forms a collection of "spiders". The internal vertices of different spiders are disjoint, and each spider has at least two legs, while every leg of a spider originates at a distinct terminal. Notice that in general it is not hard to define prefix $p(f)$ for each path f so that the set of prefixes forms a collection of internally disjoint spiders, as long as we allow one-legged spiders. However it is crucial for us that the spiders have at least two legs, since we use them to connect terminals to one another. This requirement makes the task of prefix decomposition more challenging, and it is not even clear a priori why such a decomposition should exist. The resulting spiders provide a convenient way of connecting the terminals to one another, which can be thought of as re-routing the flow from a terminal $t \in T$ to s via other terminals. Since the spiders do not share any edges, the cost of this re-routing is close to the cost of the optimal solution.

This approach allows us to obtain an $O(k \log n)$-approximation for the edge cost version. In order to obtain an approximation algorithm for the vertex cost version, we start with the prefix decomposition theorem again. However, since the costs are now on vertices, we cannot use the spiders to construct the k-tuples of paths connecting each terminal t to $(T \cup \{s\}) \setminus \{t\}$. The reason is that a spider may have many legs, and each path re-routed through the spider will use the vertex that serves as the spider head. Therefore, if we find the k-tuple of paths connecting each terminal t to $(T \cup \{s\}) \setminus \{t\}$ separately, we may end up paying very high cost. Instead, the spider decomposition theorem allows us to write a strong linear program. We then round a fractional solution to this LP to obtain the desired collection of vertices that k-vertex connects each terminal t to $(T \cup \{s\}) \setminus \{t\}$, and whose cost is close to OPT.

Organization: Section 2 formally defines the single-source k-vertex connectivity problem, and introduces some notation that we use throughout the paper. In Sections 3 and 5, we study the problem in the edge cost model and vertex cost model, establishing Theorems 1 and 2, respectively. The proof of both these results relies on a structural result that we refer to as the Prefix Decomposition theorem, and establish in Section 4. We conclude with some future directions in Section 6.

2. Preliminaries

In the *single-source k-vertex connectivity* problem, we are given a graph $G(V, E)$, a source s, a subset $T \subseteq V$ of terminals, and an integer k. In the *edge-cost version*, we are given a cost function c on edges, and the goal is to find a minimum cost subset E' of edges such that each terminal t is k-vertex connected to s in the graph induced by the edges in E'. In the *vertex-cost version*, we are given a cost function c on vertices, and the goal is to find a minimum cost subset V' of vertices, such that in the graph induced by V' every terminal t is k-vertex connected to source s. For any subset T' of terminals, we denote by $T'^{+} = T' \cup \{s\}$.

Let $T' \subseteq T$ be any subset of terminals. We say that a terminal t is *weakly k-connected* to set $T'^{+} \setminus \{t\}$ if there exist k *internally* vertex-disjoint paths from t to $T'^{+} \setminus \{t\}$. We say that terminal t is *strongly k-connected* to set $T'^{+} \setminus \{t\}$ if there exist k *internally* vertex-disjoint paths from t to $T'^{+} \setminus \{t\}$ such that each terminal in $(T' \setminus \{t\})$ is an end-

point of at most one such path. Note that if t is strongly k-connected to $(T' \setminus \{t\}) \cup \{s\}$, then if we delete any subset $X \subseteq V \setminus \{s,t\}$ of size at most $(k-1)$, terminal t remains connected by a path to some vertex in $T' \cup \{s\}$.

The cost function c can be naturally extended to subsets of edges (vertices) in the edge cost (vertex cost) model. Moreover, given a collection of paths P, slightly abusing the notation, we will use $\mathrm{c}(P)$ to denote the sum of the costs of edges (vertices) on P in edge cost (vertex cost) model. Finally, we will use OPT to denote both an optimal solution as well as its cost.

3. Single-Source VC-SNDP with Edge Costs

In this section we sketch the proof of Theorem 1. Our algorithm and its analysis are based on establishing the following property. For each terminal t, let E_t be *any* minimum cost subset of edges, such that in the graph induced by E_t, t is strongly k-connected to the set $T^+ \setminus \{t\}$. Then $\sum_{t \in T} \mathrm{c}(E_t) = O(k \cdot \mathrm{OPT})$. Note that the bound on $\sum_{t \in T} \mathrm{c}(E_t)$ holds without accounting for any sharing of edges among the solutions that strongly k-connect the terminals. We will show that this *separability property* naturally lends itself to a recursive algorithm for solving the single-source k-vertex connectivity problem.

3.1. Algorithm Description

The algorithm consists of the following three steps:

1. If $|T| \leq 10k$: for each terminal $t \in T$, find a minimum cost subset $E_t \subseteq E$ of edges, such that t is k-vertex connected to the source s in the graph $G_t = (V, E_t)$. Stop and output $\cup_{t \in T} E_t$. Otherwise: for each terminal $t \in T$, find a minimum cost subset $E_t \subseteq E$ of edges, such that t is strongly k-connected to the set $T^+ \setminus \{t\}$ in the graph $G_t = (V, E_t)$. Let $B(t)$ denote the set of k vertex disjoint paths strongly connecting t to $T^+ \setminus \{t\}$ realized by the edges in E_t, and let $\Gamma = \sum_{t \in T} \mathrm{c}(E_t)$.

2. Identify a subset $T' \subseteq T$ of size $\left\lceil \frac{|T|}{4(k+1)} \right\rceil$ such that (a) for each $t \in T'$, the paths in $B(t)$ terminate only on vertices in $T^+ \setminus T'$, and (b) $\sum_{t \in T'} \mathrm{c}(E_t) \leq \Gamma/2k$. Define $E' = \cup_{t \in T'} E_t$.

3. Recursively solve the problem on the set $T'' = T \setminus T'$ of terminals. Let E'' be the set of edges in the recursive solution. Output $E' \cup E''$ as the final solution.

Step (1) can be implemented by performing a standard min-cost max flow computation for each terminal t. Step (3) is a straightforward recursive computation. We now describe the implementation of step (2). We construct a graph $H(V_H, E_H)$ with a vertex v_t for each terminal t. A vertex v_t is connected in H to all vertices $v_{t'}$ such that a path in $B(t)$ ends at terminal t'. It is readily seen that for any subset $U \subseteq V_H$, the graph induced by vertices in U contains a vertex of degree at most $2k$. This allows us to compute in polynomial-time a coloring of H with $2k+1$ colors. We now mark all vertices v_t such that $\mathrm{c}(E_t) \leq \frac{2\Gamma}{|T|}$. By the pigeonhole principle, there exists a color class that contains at least $\frac{|T|}{4(k+1)}$ marked vertices. Let T' be an arbitrary subset of $\left\lceil \frac{|T|}{4(k+1)} \right\rceil$ terminals corresponding to the marked vertices in this color class. Clearly, $\sum_{t \in T'} \mathrm{c}(E_t) \leq \Gamma/2k$.

3.2. Cost and Feasibility Analysis

A straightforward induction on the recursion depth allows us to prove the following lemma.

Lemma 3.1 *The algorithm outputs a feasible solution for the single-source k-vertex connectivity.*

Proof: We prove the lemma by induction on the recursion depth. Clearly, when $|T| \leq 10k$, the output of the algorithm is a feasible solution for the set T of terminal. Assume now that $|T| > 10k$, and that E'' is a feasible solution for the instance defined by the set T'' of terminals. We show that $E' \cup E''$ is a feasible solution for T. Consider some terminal $t \in T$. If $t \in T''$, then by the induction hypothesis, t is k-vertex connected to s in the graph induced by E''. Assume now that $t \in T'$. If t is not k-vertex connected to s in the graph $G^* = (V, E' \cup E'')$, there exists a subset $X \subseteq V \setminus \{s,t\}$ of $(k-1)$ vertices whose removal from G^* separates t from s. However, since t is strongly k-vertex connected to $T'' \cup \{s\}$ in graph induced by E', it remains connected to some $t' \in T'' \cup \{s\}$ even after X is removed from G^*. If $t' = s$ we are done. Otherwise, $t' \in T''$, and by induction hypothesis t' is k-vertex connected to s in G^*. Therefore, even when X is removed from G^*, t remains connected to t' and t' remains connected to s in G^*. A contradiction. □

We will show that the cost Γ in Step (1) of the algorithm is $O(k) \cdot \mathrm{OPT}$. Assuming this property, we can bound the cost of the solution produced by the algorithm using the recurrence: $\alpha(p) \leq \alpha\left(p - \frac{p}{4(k+1)}\right) + O(\mathrm{OPT})$, where $\alpha(p)$ denotes the worst-case cost of a solution output by our algorithm when given as input a subset $T' \subseteq T$ of p terminals. It is easy to verify that $\alpha(|T|) = O(k \log |T|) \cdot \mathrm{OPT}$. The remainder of this section is devoted to proving that $\Gamma = O(k) \cdot \mathrm{OPT}$. The main ingredient of our proof is a Prefix Decomposition Theorem that may be of independent interest. We need the following definitions.

Definition 1 (Canonical Spider) *Let $\mathcal{M}$ be any collection of simple paths, such that each path $p \in \mathcal{M}$ has a distinguished endpoint $t(p)$, and the other endpoint is denoted by*

$v(p)$. *We say that paths in $\mathcal{M}$ form a* canonical spider *iff $|\mathcal{M}| > 1$ and there is a vertex v, such that for all $p \in \mathcal{M}$, $v(p) = v$. Moreover, the only vertex that appears on more than one path of $\mathcal{M}$ is v. We refer to v as the* head *of the spider, and the paths of $\mathcal{M}$ are called the* legs *of the spider.*

Definition 2 (Canonical Cycle) *Let $\mathcal{M} = \{g_1, \ldots, g_h\}$ be any collection of simple paths, where each path g_i has a distinguished endpoint $t(g_i)$ that does not appear on any other path in $\mathcal{M}$, and the other endpoint is denoted by $v(g_i)$. We say that paths of $\mathcal{M}$ form a* canonical cycle*, iff (a) h is an odd integer, (b) for every path g_i, $1 \leq i \leq h$, there is a vertex $v'(g_i)$ such that $v'(g_i) = v(g_{i-1})$ (here we use the convention that $g_0 = g_h$), and (c) no vertex of g_i appears on any other path of $\mathcal{M}$, except for $v'(g_i)$ that belongs to g_{i-1} only and $v(g_i)$ that belongs to g_{i+1} only (see Figure 1).*

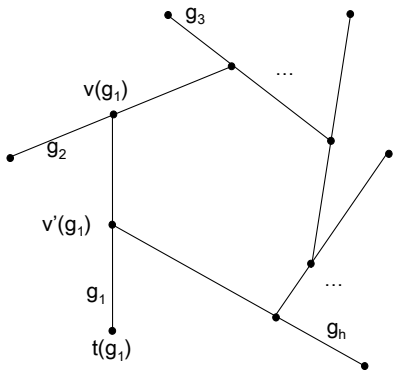

Figure 1. A canonical cycle

Assume now that we are given any collection $\mathcal{P}$ of paths, where every path $f \in \mathcal{P}$ has a distinguished endpoint $t(f)$. For $f \in \mathcal{P}$, we say that a sub-path p of f is a prefix of f iff $t(f) \in p$.

Theorem 4 (Prefix Decomposition) *Given any collection $\mathcal{P}$ of paths, where every path $f \in \mathcal{P}$ has a distinguished endpoint $t(f)$ that does not appear on any other path of $\mathcal{P}$, we can find, in polynomial time, for each path $f \in \mathcal{P}$, a prefix $p(f)$, such that in the graph induced by $\{p(f) \mid f \in \mathcal{P}\}$, the prefixes appearing in each connected component either form a canonical spider, a canonical cycle, or the connected component contains exactly one prefix $p(f)$, where $p(f) = f$ for some $f \in \mathcal{P}$.*

We defer the proof of the Prefix Decomposition Section to next section, and we proceed to show that this theorem implies that $\Gamma = O(k \cdot \text{OPT})$.

Fix an optimal solution OPT. For each $t \in T$, fix a k-tuple of internally vertex-disjoint paths $B(t)$ that connects t to s in OPT. We now define k-tuple $B'(t)$ of paths as follows. For each $f \in B(t)$, if f contains any terminal as intermediate vertex, let $t'(f)$ be the first such terminal, and otherwise let $t'(f) = s$. In order to obtain $B'(t)$, we replace each path $f \in B(t)$ by its sub-path that starts at t and ends at $t'(f)$. Let $\varphi^*(t) = \{t'(f) \mid f \in B'(t), t'(f) \neq s\}$ (it is possible that $\varphi^*(t) = \emptyset$). Notice that now no terminals appear as intermediate vertices on paths $\bigcup_{t \in T} B'(t)$, and each path in $B'(t)$ terminates at a vertex in $\varphi^*(t) \cup \{s\}$. For any terminal $t \in T$ and path $f \in B'(t)$, we say that p is a *prefix of f* iff p is a sub-path of f containing t. We say that all paths $f \in B'(t)$ belong to t. If f belongs to terminal t and p is a prefix of f, we say that p belongs to t as well.

Using the Prefix Decomposition Theorem, we can obtain the following theorem and its corollary that shows $\Gamma = O(k \cdot \text{OPT})$. The proofs are deferred to the full version [9] due to space limitations.

Theorem 5 (Weak k-connectivity) *Let $T' \subseteq T$ be any subset of terminals, such that for each $t \in T'$, $T' \cap \varphi^*(t) = \emptyset$. For each $t \in T'$, let H_t be a minimum-cost k-tuple of internally vertex disjoint paths that do not contain terminals as intermediate vertices, where each path $f \in H_t$ connects t to a vertex in $(T' \cup \varphi^*(t) \cup \{s\}) \setminus \{t\}$, and where each terminal in $\varphi^*(t)$ is an end-point of at most one path of H_t. Let E_t be the set of edges appearing on paths of H_t. Then $\sum_{t \in T} c(E_t) \leq 2OPT$.*

Corollary 1 (Strong k-connectivity) *For each $t \in T$, let E_t be a minimum cost subset of edges in G that ensures that t is* **strongly** *k-connected to the set $T^+ \setminus \{t\}$. Then $\sum_{t \in T} c(E_t) \leq O(k \cdot OPT)$.*

4. Proof of the Prefix Decomposition Theorem

We now prove Theorem 4. Note that the Prefix Decomposition theorem is a structural result, independent of the underlying cost model (that is edge or vertex costs). We will thus also utilize this result in designing an algorithm for the vertex costs case.

We start with the set $\mathcal{P}$ of paths, where each path $f \in \mathcal{P}$ has one distinguished endpoint $t(f)$ that does not appear on any other paths in $\mathcal{P}$, and the other endpoint is denoted by $z^*(f)$. We will view path f as starting at $t(f)$ and ending at $z^*(f)$. Let $V^* = \{z^*(f) \mid f \in \mathcal{P}\}$. Throughout the algorithm we will define prefixes of paths in $\mathcal{P}$. Prefix of a path f is denoted by $p(f)$, and it is defined to be the portion of f between $t(f)$ and some other vertex $z(f) \in f$. At the beginning of the algorithm, $z(f) = z^*(f)$ for all f. Throughout the algorithm, path f is trimmed by moving $z(f)$ closer to $t(f)$. We can trim a path several times, and during the execution of the algorithm the prefix may only become shorter.

Definition 3 (Canonical Set) *Let $\mathcal{P}' \subseteq \mathcal{P}$ be any set of paths. A set of prefixes $p(f)$ for $f \in \mathcal{P}'$ is said to form a* canonical set *iff in the graph induced by the prefixes $\{p(f) \mid f \in \mathcal{P}'\}$, each connected component is either a canonical spider, a canonical cycle, or it contains a single prefix $p(f) = f$ for some $f \in \mathcal{P}$.*

In order to complete the proof of the Prefix Decomposition theorem, it is enough to show that we can define, for each path $f \in \mathcal{P}$ a prefix $p(f)$, such that the set of prefixes of paths in $\mathcal{P}$ forms a canonical set.

We show an algorithm that finds such prefixes $p(f)$. We start with the set $\mathcal{P}$ of paths and prefixes $p(f) = f$ for all $f \in \mathcal{P}$. Throughout the algorithm we will maintain a collection D of *dead paths*, and all other paths in $\mathcal{P}$ are referred to as *live paths*. If f is a live path, then we refer to $p(f)$ as a *live prefix*. At the beginning, D contains a path $f \in \mathcal{P}$ iff none of the vertices of f belongs to other paths in $\mathcal{P}$. Clearly, prefixes in D form a canonical set.

Definition 4 (Special Vertices) *For any live path f, we say that u is a special vertex of $p(f)$ if u belongs to another live prefix. The i_{th} special vertex of f (counting from $t(f)$), is denoted by $u_i(f)$.*

Note that multiple special vertices on f might result due to intersections of f with another path f', so $u_i(f)$ and $u_j(f)$ may be a result of the intersection with the same path. We maintain the following invariants throughout the algorithm:

C1. The set of path prefixes in D forms a canonical set.

We define a set U of vertices as follows. For each connected component $\mathcal{P}'$ of D, if $\mathcal{P}'$ is a canonical spider, then U contains its head, and if $\mathcal{P}'$ contains only one prefix $p(f) = f$ for some $f \in \mathcal{P}$, then U contains $z^*(f)$.

C2. The only vertices that dead paths and live paths may share are vertices in U. For any live path f, its prefix $p(f)$ may contain at most one vertex in U. If $p(f)$ contains a vertex of U, then this vertex must be $z(f)$, the last vertex of $p(f)$.

C3. For a live path f, one of the following conditions must hold:

- $p(f)$ contains a vertex in U (and this is the last vertex of $p(f)$).
- $p(f)$ does not contain any vertex in U but $z(f) = z^*(f)$.
- $p(f)$ neither contains a vertex in U nor is $z(f) = z^*(f)$, but there is another live path $f' \neq f$ such that $z(f)$ is the first special vertex of f'. We refer to f' as a *witness* for f, denoted by $f' = W(f)$. Note that $W(f)$ is defined only if the first two conditions do not hold.

The algorithm works in iterations. In each iteration, we either trim prefixes of some live paths, or move some live paths to set D. At the beginning of the algorithm, it is easy to see that properties C1–C3 hold. Assume these properties hold before the current iteration. We show how to perform an iteration and maintain the properties. The algorithm ends when $D = \mathcal{P}$. It will be clear from the description given below that each iteration can be executed in polynomial time. We will use the following easy observation in our analysis:

Proposition 1 *Assume we have a collection of paths D, and prefixes of paths in $\mathcal{P}$ for which properties C1–C3 hold. Let f be any live path, $v \in p(f)$ be any special vertex of f, and f' be a live path whose first special vertex is v. Then if we trim f at vertex v by setting $z(f) = v$, and either set $W(f) = f'$ or add v to U, property C3 continues to hold.*

Proof: Note that no assertion is being made about maintaining properties C1 or C2. The only potential problem is paths f^* for which $W(f^*) = f$. In this case $p(f^*)$ must contain $u_1(f)$, the first special vertex of f. But since we trim f at its special vertex and do not trim f^*, $u_1(f)$ continues to be the first special vertex of $p(f)$, that belongs to $p(f^*)$. □

Iteration Description: We now describe how an iteration of the algorithm is executed. It will be clear from the description given below that each iteration can be executed in polynomial time.

At the beginning of an iteration, each live path f marks its first special vertex $u_1(f)$. If no such vertex exists, then f marks the vertex $z(f)$. Note that if $p(f)$ does not contain any special vertices then $z(f) = z^*(f)$ or $z(f) \in U$ must hold by Invariant C3. We then perform one of the next five steps. We consider the steps in the order in which they appear, and perform the earliest listed step that is applicable. After an applicable step is executed, the iteration ends and we proceed to the next iteration. As long as the set of live paths is non-empty, we will show that one of the steps below necessarily applies.

Step 1 Is performed if some path f marks its vertex $z(f)$, and either $z(f) \in U$ or $z(f) = z^*(f)$. We add f to D, where it either becomes part of an existing canonical spider or defines a new connected component.

- If $z(f) \in U$, then $p(f)$ becomes part of an existing canonical spider. Notice that $p(f)$ does not share any vertices (except for $z(f)$) with another live prefix, and therefore, properties C1 and C2 still hold. It is impossible that $f = W(f')$ for some live path f', since the only vertex that f shares with any other live prefix is $z(f)$, which belongs to U. Therefore, property C3 is still true.

- If $z(f) = z^*(f) \notin U$, then f defines a new connected component in D, and we add $z^*(f)$ to U. Since $z(f) = z^*(f)$, we have that $p(f) = f$. It is easy to see that properties C1–C3 still hold.

The case when some path f marks a vertex $z(f)$, and neither $z(f) \in U$ nor $z(f) = z^*(f)$ (and therefore, $z(f)$ is a special vertex for f), will be handled as one of the cases in Step 2. Notice that in this case, by property C3, $W(f)$ must exist, and both f and $W(f)$ mark the same vertex $z(f)$.

Step 2 Is performed if there are two or more paths that marked the same vertex v. Let $F = \{f_1, \ldots, f_h\}$ be the set of all paths that marked v, and let F' be the collection of all other live paths whose prefixes contain v (possibly $F' = \emptyset$). We perform the following actions: (i) the prefixes of all paths in F and F' are trimmed at v, (ii) the paths in F are added to set D of dead paths, and (iii) the vertex v is added to the set U. Notice that prefixes of paths of F form a canonical spider, and the only vertex they share with any live prefixes is v, while they do not share any vertex with dead prefixes. It is thus easy to see that properties C1 and C2 still hold. We can use Proposition 1 to prove that property C3 is also still true (we can think about this process as trimming paths in F and F' one by one and using Proposition 1 after each step to show that property C3 is maintained).

Step 3 Is performed if there are two live paths f, f', and f' marked some vertex v that lies strictly between $t(f)$ and $z(f)$ on $p(f)$. We then trim path f at v by setting $z(f) = v$ and $W(f) = f'$. Clearly, properties C1–C2 still hold, and we can invoke Proposition 1 to see that property C3 is still true (as v must be a special vertex of f).

Assume now that we have been unable to perform any of the Steps 1 through 3. This means that for each path f', if v is the vertex that f' has marked, then $v \neq z(f')$ and v is the last vertex on some other path f. We will say that f' gives its token to f. If there are several paths containing v, then f' gives its token to all these paths.

Proposition 2 *If none of the Steps 1 through 3 is applicable, then the token distribution results in each live path giving and receiving exactly one token.*

Proof: From the description above, it is clear that each path gives at least one token. We will argue that no path can receive more than one token. This clearly implies that every path gives and receives exactly one token. Suppose there exists a path f that receives two or more tokens. Since Step 3 is not applicable, two or more paths must have marked $z(f)$. But then Step 2 is applicable, a contradiction. □

Proposition 3 *If none of the Steps 1 through 3 is applicable, then (a) the prefix of each live path f has at least two special vertices $u_1(f)$ and $u_2(f)$, (b) no live prefix contains a vertex from the set U, and (c) if f' gives token to f and $z(f) \neq z^*(f)$ then $W(f) = f'$.*

Proof: By Proposition 2, we know that each live path f gives a token by marking the vertex $u_1(f)$. Since each live path f also receives a token, some path f' must have marked a vertex u that lies on $p(f)$. If $u = u_1(f)$ then Step 2 would have been applicable. So u must be distinct from $u_1(f)$. Hence $u_2(f)$ must exist.

To see that no live prefix contains a vertex of U, note that if a path f contains a vertex of U, then it must be $z(f)$ by Property C2. Since $z(f)$ has been marked by path $f' = W(f)$, and $z(f) \in U$, it follows that $z(f) = z(f')$. We should have then executed Step 1.

By Proposition 2, for each live path f, there is a unique live path f' that gives a token to f. Hence f' is the unique live path that contains $z(f)$ as its first special vertex. It follows that $W(f) = f'$. □

Step 4 Is performed if for some f, the path f' to which f gave its token also contains $u_2(f)$. This means that f' meets f at least twice: first at $u_2(f)$ then at $u_1(f)$. We can then trim f' at $u_2(f)$ by setting $z(f') = u_2(f)$. We set $W(f') = f$ as before. It is easy to see that all properties continue to hold, since now $u_2(f)$ becomes the first special vertex of f.

Assume that none of the Steps 1 through 4 are applicable. We create a directed graph where for every live path f, there is a vertex w_f. We connect w_f to $w_{f'}$ iff the path f^* that gave its token to f has $u_2(f^*)$ belonging to f'. Notice that the out-degree of every vertex in this graph is at least 1, and since Step 4 was not applicable, no self-loops can exist. Therefore, we can find a simple cycle in this graph. Assume that the paths corresponding to the vertices on the cycle are $f_1, \ldots, f_h$ (we use the convention that $f_{h+1} = f_1$). For each $i : 1 \leq i \leq h$, let f'_i be the path that gave its token to f_i. Then $u_2(f'_i)$ belongs to f_{i+1} (see Figure 2). We will need the following simple observation.

Proposition 4 *For each $i : 1 \leq i \leq h$, $u_2(f'_i)$ lies strictly between $t(f_{i+1})$ and $z(f_{i+1})$.*

Proof: Assume for contradiction that $u_2(f'_i) = z(f_{i+1})$ (it is impossible that $u_2(f'_i) = t(f_{i+1})$). Then path f'_{i+1}, which has marked $z(f_{i+1})$ has two paths intersecting it at $z(f_{i+1})$, which is impossible since every path gives and receives exactly one token. □

We say that path f_i is bad iff for some $j : 1 \leq j \leq h$, $f_i = f'_j$, and $u_2(f'_{i-1}) = u_1(f_i)$. If no bad paths exist in

our cycle, we perform Step 5.

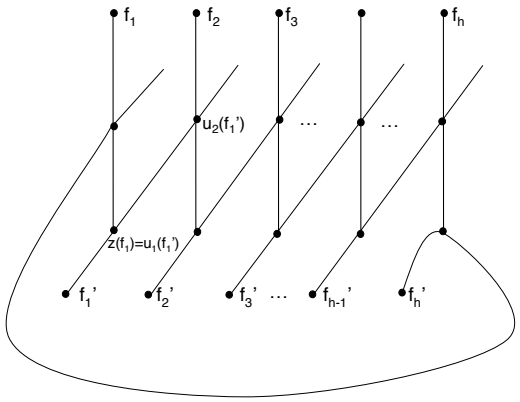

Figure 2. Before the execution of step (5).

Step 5 We trim each path f_i, $1 \leq i \leq h$, by setting $z(f_i) = u_2(f'_{i-1})$ and setting $W(f_i) = f'_{i-1}$ (See Figure 3). It is easy to see that properties C1 and C2 continue to hold. We now focus on showing that property C3 still holds as well. We only perform trimming of paths $\{f_i\}_{i=1}^{h}$. So we only need to take care of two cases:

- For each path $f_i \in \{f_1, \ldots, f_h\}$, we need to show that after the trimming f_i contains the first special vertex of $W(f_i) = f'_{i-1}$, and this is the last vertex $z(f_i)$ of $p(f_i)$.
- If $f \notin \{f_1, \ldots, f_h\}$, and $W(f) = f_i$, then we need to show that f contains the first special vertex of f_i, and it is the last vertex of $p(f)$.

For the second case, observe that f_i has been trimmed on its special vertex, and f has not been trimmed in this iteration. Therefore, we can use arguments similar to Proposition 1 to show that f still contains the first special vertex of f_i.

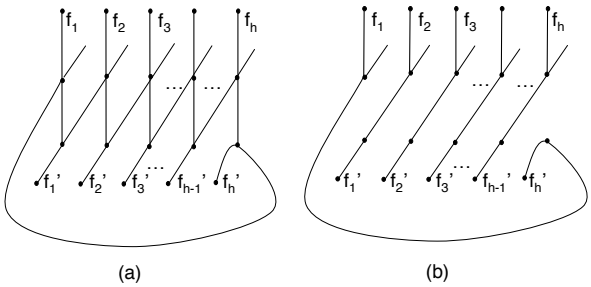

Figure 3. Illustration of step (5)

Consider now the first case. Recall that from Proposition 4, the trimming of f_{i-1} made it shorter by at least one edge. Therefore, after the trimming, f_{i-1} does not contain $u_1(f'_{i-1})$, which stops being the first special vertex of f'_{i-1}. (Recall that f_{i-1} and f'_{i-1} are the only paths that contained $u_1(f'_{i-1})$, since otherwise f'_{i-1} gives at least two tokens). If we show that $u_2(f'_{i-1})$ continues to belong to f'_{i-1} after the trimming step, then it now becomes the first special vertex of f'_{i-1}. Vertex $u_2(f'_{i-1})$ belongs to f_i even after trimming, and is the last vertex of $p(f_i)$. We only need to show that $u_2(f'_{i-1})$ still belongs to f'_{i-1} after the trimming. The only way f'_{i-1} has been trimmed is if $f'_{i-1} = f_j$ for some j. If $u_2(f'_{i-1})$ does not belong to the new prefix of $f'_{i-1} = f_j$ is then $u_2(f'_{j-1})$, at which f_j has been trimmed lies before $u_2(f_j)$ on its prefix. But since $u_2(f'_{j-1})$ is a special vertex of f_j, it must be that $u_2(f'_{j-1}) = u_1(f_j)$, and since $f_j = f'_{i-1}$, we have that f_j is a bad path, a contradiction.

Step 6 If none of the above steps has been performed, then there is a bad path $f = f_i = f'_j$ whose vertex w_f belongs to the cycle. We prove that in this case, every path f_j is bad, and a subset of prefixes of $\{f_1, \ldots, f_h\} = \{f'_1, \ldots, f'_h\}$, form a canonical cycle that can be added to the set D. We start with the following claim:

Claim 1 *If $f_i = f'_j$ is bad, then f_j is also bad and $f_j = f'_{i-1}$.*

Proof: Let $g = f'_{i-1}$. Since f_i is bad, its first special vertex is the second special vertex of g. Therefore, f_i gave its token to g, and the prefix of g currently ends at $z(g) = u_1(f)$. Thus, g has exactly two special vertices: $u_1(g)$ and $u_2(g) = z(g)$. Since $f_i = f'_j$, and f_j is the path to which f'_j gave its token, it follows that $f_j = g = f'_{i-1}$. Consider now f'_{j-1}. Its second special vertex $u = u_2(f'_{j-1})$ belongs to g and must be a special vertex of g. From Proposition 4, it cannot be the last vertex of g, $z(g) = u_2(g)$. So it must be $u_1(g)$. It follows that $g = f_j = f'_{i-1}$ is also bad. □

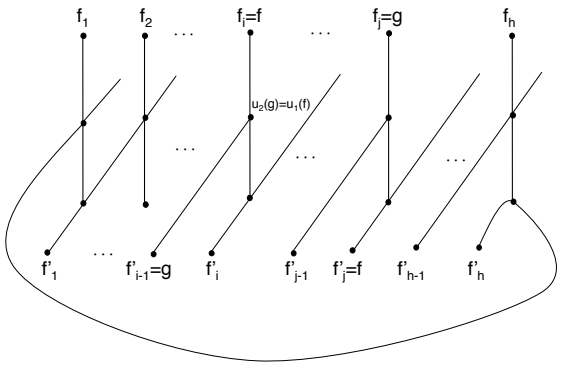

Figure 4. Proof of claim 1

Corollary 2 *If $f_i = f'_j$ is bad, then f_{i-1} is bad and $f_{i-1} = f'_{j-1}$.*

Proof: Assume that $f_i = f'_j$ is bad. Then from Claim 1, f_j is bad and $f_j = f'_{i-1}$. We now apply the same claim again to f_j serving as f_i and f'_{i-1} serving as f'_j. We get that f_{i-1} is also bad and $f_{i-1} = f'_{j-1}$. □

From the above corollary, all paths $F = \{f_1, \ldots, f_h\}$ are bad, and $F = \{f'_1, \ldots, f'_h\}$. If $f_i = f'_j$ is bad, then since f_{j+1} is also bad, the second special vertex of $f_i = f'_j$ is the first special vertex of f_i, and so it is also the last vertex of $p(f_i)$. It follows that every prefix $p(f_i)$, for $f_i \in F$, contains exactly two special vertices: $v_1(f_i)$ and $v_2(f_i) = z(f_i)$. Moreover, $v_1(f_i) = v_2(f'_{i-1})$ and no other live path contains it; $v_2(f_i) = v_1(f'_i)$ and no other live path contains it. Let $h' : 1 \leq i \leq h$ be such that $f_1 = f'_{h'}$. Considered the ordered set of prefixes $\mathcal{M} = (p(f_1), p(f'_1), p(f_2), p(f'_2), \ldots, p(f'_{h'-1}), p(f_{h'}))$. Notice that the number of prefixes in $\mathcal{M}$ is odd. From the above discussion, prefixes in $\mathcal{M}$ form a canonical cycle. Moreover, vertices appearing on the prefixes in $\mathcal{M}$ do not appear on any other live prefixes and do not belong to paths in D. We move the paths whose prefixes belong to $\mathcal{M}$ to D.

5. Single-Source VC-SNDP with Vertex Costs

We now consider the case when the costs are on vertices. We are given a graph $G = (V, E)$ with cost $\mathrm{c}(v) \geq 0$ for each vertex $v \in V$, a subset $T \subseteq V$ of terminals, and a source $s \in V \setminus T$. The goal is to find a minimum cost subset $V' \subseteq V$ of vertices, such that in the graph induced by V' every terminal is k-vertex connected to s. For each subset $T' \subseteq T$ of terminals, we again denote by T'^{+} the set $T' \cup \{s\}$ of vertices. We assume w.l.o.g. that the cost of every vertex in T^+ is 0 since any solution must include them. The main theorem of this section is the following.

Theorem 6 *Let $G = (V, E)$ be any instance of single-source k-vertex connectivity problem with terminal set T, source s and vertex costs c. Given any subset $T' \subseteq T$ of terminals, there is a randomized polynomial time algorithm that finds, with high probability, a subset $V' \subseteq V$ of vertices of cost $O(\mathit{OPT} \cdot k \log n)$ with the following properties. The graph induced by V' contains, for each $t \in T'$, a k-tuple $F(t)$ of internally vertex disjoint paths. Each path in $F(t)$ connects t to some vertex in $T^+ \setminus \{t\}$, while for terminals $t' \in T \setminus T'$ at most one path in $F(t)$ terminates at t'. Moreover, paths in $F(t)$ do not contain any terminals of T as intermediate vertices.*

The theorem above can be used to obtain the following analogue of Corollary 1.

Corollary 3 *There is a randomized polynomial time algorithm, that finds, with high probability, a subset $V' \subseteq V$ of vertices of cost $O(\mathit{OPT} \cdot k^6 \cdot \log n)$, such that in the graph induced by V' every terminal $t \in T$ is strongly k-vertex connected to $T^+ \setminus \{t\}$.*

We now state our algorithm for the vertex costs case.

1. If $|T| \leq 10k$: for each terminal $t \in T$, find a minimum cost subset $V_t \subseteq V$ of vertices, such that t is k-vertex connected to the source s in the graph induced by V_t. Stop and output $\cup_{t \in T} V_t$. Otherwise, using Corollary 3, find a set V' of vertices of cost $O(\mathrm{OPT} \cdot k^6 \cdot \log n)$, such that in the graph induced by V' every terminal $t \in T$ is strongly k-vertex connected to $T^+ \setminus \{t\}$. Let $F(t)$ denote the k-tuple of paths that strongly k-vertex connect t to $T^+ \setminus \{t\}$.
2. Identify a subset $T' \subseteq T$ of size $\left\lceil \frac{|T|}{2(k+1)} \right\rceil$ such that for each $t \in T'$, the paths in $F(t)$ terminate only on vertices in $T^+ \setminus T'$.
3. Recursively solve the problem on the set $T'' = T \setminus T'$ of terminals. Let V'' be the set of vertices in the recursive solution. Output $V' \cup V''$ as the final solution.

Using similar arguments as in Lemma 3.1 it is easy to see that the algorithm outputs a feasible solution. The total depth of recursion is bounded by $O(k \log n)$ and thus the solution cost is at most $O(\mathrm{OPT} \cdot k^7 \log^2 n)$.

The proof of Theorem 6 is based on rounding an LP relaxation for computing a minimum cost spider decomposition. We defer the details of the proofs of Theorem 6 and Corollary 3 to the full version [9].

6. Concluding Remarks

The results in this paper represent progress towards closing the gaps in our understanding of the approximability of single-source k-vertex connectivity. While there still remains a separation between the upper and lower bounds in the single-source vertex connectivity, the gap in the upper and lower bounds for the general VC-SNDP is far more striking. For any fixed $k \geq 3$, the upper bound is a polynomial-ratio approximation algorithm while the lower bound is an APX-hardness. Perhaps both the approximability factor and the hardness factor can be much improved.

References

[1] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal of Computing*, 24(3):440–456, 1995.

[2] V. Auletta, Y. Dinitz, Z. Nutov, and D. Parente. A 2-approximation algorithm for finding an optimum 3-vertex-connected spanning subgraph. *Journal of Algorithms*, 32(1):21–30, 1999.

[3] M. Bern and P. Plassmann. The steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, 1989.

[4] J. Cheriyan, T. Jordan, and Z. Nutov. On rooted node-connectivity problems. *Algorithmica*, 30(3):353–375, 2001.

[5] J. Cheriyan, S. Vempala, and A. Vetta. An approximation algorithm for the minimum-cost k-vertex connected subgraph. *SIAM Journal of Computing*, 32(4):1050–1055, 2003.

[6] J. Cheriyan, S. Vempala, and A. Vetta. Network design via iterative rounding of setpair relaxations. *Combinatorica*, 26(3):255–275, 2006.

[7] T. Chakraborty, J. Chuzhoy, and S. Khanna. Network Design for Vertex Connectivity. In *Proceedings of ACM Symposium on Theory of Computing* (STOC), 2008.

[8] C. Chekuri and N. Korula. Single-sink network design with vertex connectivity Requirements. *Manuscript*, April 2008.

[9] J. Chuzhoy and S. Khanna. Algorithms for single-source vertex connectivity. Available at http://ttic.uchicago.edu/˜cjulia/ and http://www.cis.upenn.edu/˜sanjeev/.

[10] Y. Dinitz and Z. Nutov. A 3-approximation algorithm for finding optimum 4, 5-vertex-connected spanning subgraphs. *Journal of Algorithms*, 32(1):31–40, 1999.

[11] J. Fakcharoenphol and B. Laekhanukit. An $O(\log^2 k)$-approximation algorithm for the k-vertex connected subgraph problem. In *Proceedings of ACM Symposium on Theory of Computing* (STOC), 2008.

[12] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM* 45(4) pp. 634-652, 1998.

[13] L. Fleischer, K. Jain, and D. P. Williamson. Iterative rounding 2-approximation algorithms for minimum cost vertex connectivity problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006.

[14] A. Frank and T. Jordan. Minimal edge-coverings of pairs of sets. *Journal of Combinatorial Theory, Series B*, 65(1):73–110, 1995.

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

[16] M. Goemans and D. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In *Approximation Algorithms, D. Hochbaum, Ed., PWS*, 1997.

[17] M. X. Goemans, A. V. Goldberg, É. Tardos, S. A. Plotkin, D. B. Shmoys, and D. P. Williamson. Improved approximation algorithms for network design problems. In *Proceedings of the fifth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 223–232, 1994.

[18] M. X. Goemans, M. Mihail, V. Vazirani, and D. P. Williamson. A primal-dual approximation algorithm for generalized steiner network problems. *Combinatorica*, 15(3):435–454, 1995.

[19] K. Jain. Factor 2 approximation algorithm for the generalized steiner network problem. In *Proceedings of the thirty-ninth annual IEEE Foundations of Computer Science (FOCS)*, pages 448–457, 1998.

[20] S. Khuller and B. Raghavachari. Improved approximation algorithms for uniform connectivity problems. *Journal of Algorithms*, 21(2):434–450, 1996.

[21] P. N. Klein and R. Ravi. A Nearly Best-Possible Approximation Algorithm for Node-Weighted Steiner Trees. *Journal of Algorithms*, 19 (1): 104–115, 1995.

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

[23] G. Kortsarz and Z. Nutov. Approximating node connectivity problems via set covers. *Algorithmica*, 37(2):75–92, 2003.

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

[25] C. Lund, and M. Yannakakis. On the hardness of approximating minimization problems. *J.ACM 41,* 5, 960–981, 1994.

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