Improved Approximation Algorithms for Uniform Connectivity Problems

Samir Khuller *   Balaji Raghavachari †

Abstract

The problem of finding minimum weight spanning subgraphs with a given connectivity requirement is considered. The problem is NP-hard when the connectivity requirement is greater than one. Polynomial time approximation algorithms for various weighted and unweighted connectivity problems are given.

The following results are presented:

1. For the unweighted k-edge-connectivity problem an approximation algorithm that achieves a performance ratio of 1.85 is described. This is the first polynomial-time algorithm that achieves a constant less than 2, for all k.
2. For the weighted k-vertex-connectivity problem, a constant factor approximation algorithm is given assuming that the edge-weights satisfy the triangle inequality. This is the first constant factor approximation algorithm for this problem.
3. For the case of biconnectivity, with no assumptions about the weights of the edges, an algorithm that achieves a factor asymptotically approaching 2 is described. This matches the previous best bound for the corresponding edge connectivity problem.

1 Introduction

Connectivity is fundamental to the study of graphs and graph algorithms. Recently, many approximation algorithms for finding subgraphs that meet given connectivity requirements have been developed [1, 11, 14, 15, 19, 21, 22]. These results provide practical approximation algorithms for NP-hard network design problems, via an increased understanding of connectivity properties.

In this paper we focus on uniform k-connectivity problems. The term connectivity refers to both edge and vertex connectivities, unless specified. The input is an integer k, a k-connected graph $G = (V, E)$ and a weight function w on the edges of G. The goal is to find a minimum-weight k-connected subgraph of G. The problem is known to be NP-hard [12] even when the weights are all identical (i.e., the unweighted case). We present improved approximation algorithms for unweighted and weighted connectivity problems.

The practical motivation to study this problem is the following. Let G denote all feasible links of a proposed communications network. An edge $e = (a, b)$ denotes the feasibility of

*Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF Research Initiation Award CCR-9307462. E-mail : samir@cs.umd.edu.

†Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688. Research supported in part by NSF Research Initiation Award CCR-9409625. E-mail : rbk@utdallas.edu.

adding a link from site a to site b, and its weight, $w(e)$, represents the cost of constructing it. A minimum spanning tree in G is the smallest connected subgraph, i.e., the cheapest network that will allow the sites to communicate. Such a network is highly susceptible to failures, since it cannot survive the failure of even a single link or site. For more reliable communication, one desires spanning subgraphs of higher connectivities. A network of edge-connectivity (vertex-connectivity) k continues to allow communication between functioning sites even after as many as $k-1$ links (sites) have failed. The problem of finding low-cost fault-tolerant networks naturally leads to the minimum-weight k-connected spanning subgraph problem. Further applications and the importance of this problem are discussed by Grötschel, Monma and Stoer [16].

1.1 Our results

Unweighted Connectivity Results:

For this problem the previous best approximation factor was 2. This follows from the following two facts. Any minimal k-connected subgraph contains at most $k(n-k)$ edges [2]. In any k-connected graph, the degree of each vertex is at least k, thus implying a lower bound of $kn/2$ on the number of edges in any optimal solution. Efficient algorithms for finding k-connected subgraphs with at most $k(n-1)$ edges were given by Nagamochi and Ibaraki [20] and Cheriyan, Kao and Thurimella [3].

We observe that such approximation algorithms do not exploit any structural properties of an optimal solution, other than the trivial lower bound on the degree of each vertex. Since these algorithms provide an absolute upper bound on the number of edges, there is no way to improve their approximation factors. Any algorithm which obtains a solution that is provably better than 2 must exploit the structure of the problem and prove better lower bounds on an optimal solution. Previously, algorithms that obtained factors less than 2 were known only for the case $k=2$; Khuller and Vishkin [19] gave an approximation algorithm that achieved a factor of 1.5 for the edge-connectivity case, and 1.66 for the vertex connectivity case. The vertex connectivity bound was subsequently improved to 1.5 by Garg, Santosh and Singla [13]. By combining the biconnectivity algorithm [19], and the sparse certificate algorithm [3] one can easily obtain a factor of $2-1/k$ for the k-edge-connectivity problem, but this approaches 2 as k increases. Recently, Karger [17] has given an algorithm with a factor of $1+O(\sqrt{\log n/k})$ using randomized rounding of the fractional solution obtained from the corresponding linear program, together with the idea of finding maximal forests. This algorithm is useful when $k \gg \log n$.

In this paper we provide the first approximation algorithm which breaks the barrier of 2 for the unweighted k-edge-connectivity problem (for all k). We give an algorithm and a lower bounding method that yields an approximation ratio of about 1.85 for the k-edge-connectivity problem. At a high level, the structure of our algorithm is based on the method of [3], but it uses depth first search to obtain a better solution in each phase. Our algorithm is quite straight-forward, but proving a performance ratio of less than 2 for the algorithm requires a subtle analysis of the structure of any optimal solution.

Weighted Connectivity Results:

Here each edge of the input graph G has a nonnegative weight. For the k-edge-connected subgraph problem, an approximation factor of 2 was achieved by Khuller and Vishkin [19]. Approximation algorithms with constant performance ratios (for all k) are not known for the

k-vertex-connected subgraph problem. The best known algorithm to find a k-vertex-connected subgraph is due to Ravi and Williamson [21] that achieves a factor of $2H(k)$, where $H(k)$ is the kth Harmonic number ($H(k) = 1 + \frac{1}{2} \ldots + \frac{1}{k}$).

Frederickson and JáJá [8] considered the problem of computing a minimum-weight biconnected spanning subgraph. They gave an approximation algorithm for a more general graph augmentation problem and used it to obtain a 3-approximation algorithm for the biconnectivity problem. For $k = 2$, Ravi and Williamson's algorithm also achieves a ratio of 3. In this paper, we present an approximation algorithm for the minimum-weight biconnected subgraph problem with a performance ratio of $2 + 1/n$.

Not much more is known about the k-vertex-connectivity problem for the special case when the weights satisfy the triangle inequality. For $k = 2$, it is easy to show that the TSP algorithm of "doubling" the minimum spanning tree has a performance guarantee of 2. Frederickson and JáJá [9] proved that Christofides' algorithm [4] (for the TSP problem) has a performance guarantee of 1.5 for the minimum-weight biconnectivity problem as well. The analysis for the biconnectivity algorithm is more complicated since the relationship between the weight of a minimum-weight matching and the weight of an optimal biconnected subgraph is not obvious.

In this paper we give an algorithm for the k-vertex-connectivity problem when the edges satisfy the triangle inequality. The performance guarantee is at most $2 + 2(k-1)/n$, which is less than 4 for all k and asymptotically tends to 2 for fixed k, as n tends to infinity.

2 Preliminaries

A graph is said to be k-vertex-connected (or simply k-connected) if the deletion of up to $k-1$ vertices does not disconnect the remaining vertices of the graph. Analogously, a graph is k-edge-connected if the deletion of up to $k-1$ edges does not disconnect the graph. A set of paths between vertices u and v is said to be *openly disjoint* if they do not share any internal vertices. This definition is extended to the case when v is replaced by a set of vertices as follows: a set of k paths between u and a set of vertices R with $|R| = k$ is openly disjoint if the paths are all vertex-disjoint (except for sharing the end-point u), and each of the paths starts at u and ends at a distinct vertex of R. If $u \in R$ then we find $k-1$ openly disjoint paths from u to the vertices in $R - u$. We say u and v are k-edge-connected, if there exist k edge-disjoint paths between them. The same definition can be extended to vertex connectivity by replacing "edge-disjoint" with "openly disjoint." Note that k-edge-connectivity can be viewed as a binary relation between vertices. It is a transitive relation because, if a graph G contains k edge-disjoint paths between u and v, and k edge-disjoint paths between v and w, then it implies that G has k edge-disjoint paths between u and w.

3 Unweighted k-Edge-Connectivity

In the edge connectivity version of the connectivity problem, the input is an integer k and an undirected graph $G = (V, E)$ with edge connectivity at least k. The problem is to compute a k-edge-connected spanning subgraph of G using the minimum number of edges. Since the degree of any vertex must be at least k in a k-connected graph, any solution to the above problem must have at least $\lceil kn/2 \rceil$ edges. Any minimally k-edge-connected graph has at most $k(n-k)$ edges [2], and this provides a 2-approximation algorithm. Khuller and Vishkin [19] provided the first approximation algorithm for $k = 2$ with a performance ratio less than 2. Their algorithm is based on depth-first search and has a performance guarantee of 3/2. A simple generalization of their algorithm has a performance guarantee of $2 - 1/k$ for all k. In this section we provide the first algorithm for unweighted k-edge-connectivity with an approximation factor strictly less than 2 for all values of k. The structure of our algorithm is similar to the one used by Cheriyan, Kao and Thurimella [3], where the connectivity of the solution is increased in stages. The main idea behind the algorithm for finding a sparse k-edge-connected subgraph is to repeatedly find and delete a maximal spanning forest from the graph G. After this is repeated k times, the deleted edges form a k-edge-connected spanning subgraph of G [3, 20].

3.1 The algorithm

We first describe our algorithm for even values of k. The algorithm works in phases. It starts with an empty subgraph S. In each phase the edge connectivity of S is increased by 2. In $k/2$ phases the connectivity of S is k and the algorithm outputs S.

We now describe the procedure that we use to augment the connectivity of S by 2 in any phase. At the beginning of phase i, S is $2i - 2$ connected. At the end of phase i, it will be $2i$ connected. We first add a depth-first maximal spanning forest F_i of $G - S$ to S. This increases the connectivity of S to $2i - 1$. Note that each tree of F_i is implicitly rooted. Since we added a depth-first forest to S, all remaining edges of $G - S$ are back edges with respect to F_i. We then scan the edges of the depth-first forest F_i in post-order. Each time we find an edge $(u, v) \in F_i$ such that u and v can be separated by the removal of $2i - 1$ edges in S (recall that S includes F_i), we add to S a highest-going back edge e from $G - S$ which also crosses (u, v). Assume that u is the parent of v in F_i. This back edge e goes from some descendant of v (could be v itself) to a vertex with the lowest dfs number, when there is a choice of edges. It is easy to identify such an edge during the execution of depth-first-search (similar information is maintained during the computation of biconnected components in a graph [5]). This step ensures that u and v are $2i$-edge-connected in S. Therefore after all edges of F_i have been scanned, S is $2i$-edge-connected. We will show that the algorithm outputs a subgraph S whose cardinality is at most a factor $(3 + \ln 2)/2 \approx 1.85$ more than the cardinality of an optimal solution.

If k is odd, we can use the above procedure to first find a $(k - 1)$-edge-connected subgraph S of G and then finally augment S to k-connectivity by the addition of any maximal spanning forest of $G - S$.

3.2 Proof of correctness

We first show that the addition of a maximal spanning forest at the beginning of each phase of the algorithm increases the connectivity by 1. We then show that adding back edges as we do in our algorithm increases the connectivity by 1 more. The following lemma is easily established from the sparse certificate proofs of [3, 20]. We provide its proof for the sake of completeness.

Lemma 3.1 *Let $G = (V, E)$ be a graph which is at least λ-edge-connected. Let S be a $(\lambda - 1)$-edge-connected spanning subgraph of G, and let F be a maximal spanning forest in $G - S$. Then $S \cup F$ is λ-edge-connected.*

Proof. Suppose there exist vertices u and v which are not λ-edge-connected in $S \cup F$. We are given the fact that S is $(\lambda - 1)$-edge-connected and therefore any set of edges separating u and v must have at least $\lambda - 1$ edges. Let C be a set of $\lambda - 1$ edges separating u and v in $S \cup F$. Clearly $C \subseteq S$ because otherwise $C \cap S$ would be a set of edges with size strictly smaller than $\lambda - 1$, separating u and v in S. Also G is λ-edge-connected and therefore there is some edge $\{x, y\}$ in $G - S$ which crosses C. Therefore x and y are in the same connected component of $G - S$ and hence in the same tree of F. At least one edge in the path from x to y in this tree must cross the cut C. This implies that C cannot be a separating set in $S \cup F$, which is a contradiction. □

Lemma 3.2 *Let $G = (V, E)$ be a graph which is at least λ-edge-connected. Let S be a $(\lambda - 2)$-edge-connected spanning subgraph of G that is augmented to $(\lambda - 1)$-edge-connectivity by the addition of a maximal spanning forest F. Then any cut C with $\lambda - 1$ edges that separates $S \cup F$ must separate some two vertices x and y that are in the same tree of F.*

Proof. Suppose there exists a cut C with $\lambda - 1$ edges that separates $S \cup F$, but does not separate any vertices within the same tree, i.e., vertices of each tree of F are in the same component when the edges of C are deleted from $S \cup F$. Since F is a maximal spanning forest, the components of $G - S$ and F are the same. Therefore C is a separator for G, which is a contradiction because $|C| = \lambda - 1$ but G is λ-edge-connected. □

Lemma 3.3 *Let $G = (V, E)$ be a graph which is at least λ-edge-connected. Let S be a $(\lambda - 2)$-edge-connected spanning subgraph of G that is augmented to $(\lambda - 1)$-edge-connectivity by the addition of a maximal spanning forest F. If $S \cup F$ is augmented with an edge set B from $E - \{S \cup F\}$ such that each pair of adjacent vertices in F cannot be separated by $\lambda - 1$ edges in $S \cup F \cup B$. Then $S \cup F \cup B$ is λ-edge-connected.*

Proof. Consider any tree T in F. Any pair of adjacent vertices in T are λ-edge-connected in $S \cup F \cup B$. Since edge connectivity is transitive, all vertices of T are in the same λ-connected component of $S \cup F \cup B$, i. e., no two vertices of T may be separated by the removal of fewer than λ edges. By Lemma 3.2, any $\lambda - 1$ edge-separator of $S \cup F$ must separate two vertices in the same tree of F. But we have shown that vertices within the same tree of F cannot be separated by the removal of $\lambda - 1$ edges and therefore $S \cup F \cup B$ is λ-edge-connected. □

Theorem 3.4 *The algorithm outlined earlier correctly outputs a k-edge-connected spanning subgraph of G.*

Proof. The proof proceeds by induction on the number of phases of the algorithm. The induction hypothesis is that when the algorithm completes phase i, S is $2i$-edge-connected. Suppose the algorithm is entering phase i of the algorithm. By the induction hypothesis, S is $2i-2$-edge-connected (which is trivially true for the base case, $i = 1$). The algorithm first adds a depth-first maximal spanning forest F_i to S. By Lemma 3.1 this augments the connectivity of S to $2i - 1$. The algorithm then ensures that all adjacent vertices of F_i are $2i$-connected by adding back edges crossing the corresponding cuts, where necessary. Therefore by Lemma 3.2, S becomes $2i$-edge-connected, thus proving the induction step. Since the algorithm runs in $k/2$ phases, the solution S that it outputs is k-edge-connected. If k is odd, after $\lfloor k/2 \rfloor$ phases, a maximal spanning forest of $G - S$ is added to S thus making it k-edge-connected. □

3.3 Performance analysis

We now analyze that the performance guarantee of the algorithm. Consider phase i of the algorithm. It adds a depth-first maximal spanning forest F_i and then a set of back edges B_i to S. Let $f_i = |F_i|$ and $b_i = |B_i|$. Let $\mathcal{OPT}(G)$ be an optimal k-edge-connected spanning subgraph of G. The following is a trivial lower bound on $\mathcal{OPT}(G)$.

Lemma 3.5 $|\mathcal{OPT}(G)| \geq \lceil kn/2 \rceil$.

We now prove the following lower bound on the cardinality of the optimal solution which lets us obtain a better factor than 2 for our algorithm.

Lemma 3.6 $|\mathcal{OPT}(G)| \geq (k - 2i + 2)b_i$.

Proof. Consider phase i of the algorithm. Let $e = (x, y)$ be a tree edge in F_i, where x is the parent of y. We first observe that whenever we add an edge in B_i to cross a cut corresponding to e, then the connectivity of x and y at the beginning of phase i is exactly $2i - 2$ (at that point S does not contain F_i). Otherwise using a proof analogous to that of Lemma 3.1, it can be shown that the connectivity between x and y is at least $2i$ after the addition of F_i to S.

Consider a cut C with $2i - 1$ edges in $S \cup F_i$ separating x and y. Let the removal of C from $S \cup F_i$ break G into two components X and $\overline{X}$ with $x \in X$ and $y \in \overline{X}$. Observe that only one edge of F_i can cross this cut — this is because S is $2i - 2$ connected, and if two or more edges of F_i cross this cut, the size of this cut would be at least $2i$. In fact, the edge in F_i crossing C is exactly the edge (x, y). Let P_e be the set of edges in $G - S$ that cross C. In other words, P_e is the set of edges in $G - S$ that connect vertices in X to vertices in $\overline{X}$.

We first observe some simple properties about P_e. In the following discussion, we consider a vertex to be both a descendant of itself and an ancestor of itself.

Lemma 3.7 *The set P_e consists of all the edges of $G - S$ that connect descendants of y to ancestors of x.*

Proof. We know that $x \in X$ and $y \in \overline{X}$. If some descendant z of y (in F_i), belongs to X then there are at least two edges of F_i in cut C (one is the edge (x, y) and the other is some edge on the path from y to z). Thus all descendants of y are in $\overline{X}$. Similarly all vertices in the tree of y in F_i that are not descendants of y must be in X. Also, the vertices of any other tree T in F_i are entirely contained in either X or in $\overline{X}$. Therefore $\overline{X}$ is exactly the vertices of the sub-tree rooted at y together with the vertices of some of the other depth first spanning trees of F_i. All edges in $G - S$ from $\overline{X}$ to X are exactly the back edges, and the single tree edge (x, y) out of the sub-tree rooted at y and these edges clearly go to (not necessarily proper) ancestors of x. □

By the definition of P_e, every k-edge-connected spanning subgraph, and in particular $\mathcal{OPT}(G)$, must have at least $k - (2i - 2)$ edges from P_e. We will show that our selection of back edges to add to B_i using the highest-going back-edge rule ensures that the sets P_e for all $e \in B_i$ are all disjoint (i.e., form a 1-packing of cuts). Once we show the disjointness of the cuts corresponding to B_i, we complete the proof of Lemma 3.6, because $\mathcal{OPT}(G)$ must contain at least $k - (2i - 2)$ edges from each of these cuts.

We now prove that the cuts are disjoint. Consider two back edges ℓ and ℓ' in B_i that were added when the algorithm examined the tree edges $e = (u, v)$ and $e' = (x, y)$ respectively.

Suppose e is processed before e'. Recall that the algorithm processes the edges of F_i in post-order. Since F_i is a depth-first spanning forest, $G - \{S \cup F_i\}$ contains only back edges and therefore, $P_e \cap P'_e$ is empty if the tree edges are located such that none is an ancestor of the other. Suppose that tree-edge $e' = (x, y)$ lies on the path from v to the root of v's tree in F_i. If (x, y) is on the path from v to the root of v's tree (see Fig. 1), then we note that ℓ does not reach higher than y in the tree. Otherwise, the edge ℓ will be sufficient to ensure that vertices x and y cannot be separated by $2i - 1$ edges, because there are 2 paths between x and y in $F_i \cup \{\ell\}$ and at least $2i - 2$ in S. Since ℓ was chosen as an edge which goes highest in the tree, there can be no edges that cross (u, v) which also cross (x, y). In other words, in this case also $P_e \cap P'_e$ is empty. This proves Lemma 3.6. □

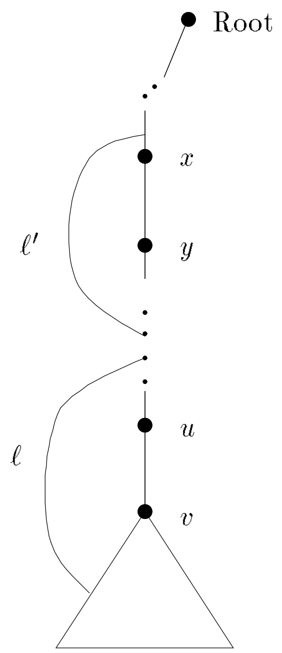

Figure 1: Structure of back edges.

Theorem 3.8 *The performance guarantee of the edge-connectivity algorithm outlined earlier is at most* $(3 + \ln 2)/2 < 1.85$.

Proof. We prove the result for even k (the proof for odd k requires minor changes, and is omitted). The algorithm adds $f_i + b_i$ edges during each phase. Since F_i and B_i are forests, $f_i < n$ and $b_i < n$. We combine the lower bounds from Lemmas 3.5 and 3.6 to obtain the performance ratio of the algorithm as follows.

$$\begin{aligned} \frac{\sum_{i=1}^{k/2}(f_i + b_i)}{|\mathcal{OPT}(G)|} &\leq \frac{3kn/4 + \sum_{i=1}^{k/4} b_i}{\max\{kn/2, \max_i\{(k - 2i + 2)b_i\}\}} \\ &\leq \frac{3}{2} + \sum_{i=1}^{k/4} \frac{1}{k - 2i + 2} \end{aligned}$$

$$
\begin{aligned}
&\leq \frac{3}{2} + \frac{1}{2}\sum_{i=1}^{k/4} \frac{1}{k/2 - i + 1} \\
&\leq \frac{3}{2} + \frac{\ln 2}{2} \quad < \quad 1.85.
\end{aligned}
$$

□

Remark: For small values of k this actually yields better factors. For example, the ratio is 1.66 for $k = 3$, 1.75 for $k = 4$, and 1.733 for $k = 5$.

4 Weighted Vertex Connectivity

Recently, Ravi and Williamson [21] gave an algorithm that finds an approximation to a minimum-weight k-vertex-connected subgraph. Their algorithm works for all k, and its performance ratio is 3 for the biconnectivity problem ($k = 2$). Their algorithm uses techniques of linear programming and it constructs the solution in stages, augmenting the connectivity by 1 in each stage. The performance guarantee for general k is $2H(k)$, where $H(k) = \sum_{i=1}^{k} 1/i$ is the kth Harmonic number.

On a related problem in directed graphs, Frank and Tardos [7] extended a technique discovered by Edmonds [6] and showed that the following problem can be solved in polynomial time. The input is a directed graph D with nonnegative weights on the edges, a root vertex r and an integer λ. The problem is to find a minimum-weight directed subgraph H of D, such that for each vertex v there are λ openly disjoint paths from r to v in the subgraph H. Gabow [10] introduced techniques for representing crossing set families based on separators, and showed that this representation scheme can be used to speed up several algorithms. His algorithm for solving the above problem of Frank and Tardos runs in $O(\lambda^2 n^2 m)$ time, where D has n vertices and m edges.

4.1 Basic technique

We give an algorithm for undirected graphs that uses the algorithm of Frank and Tardos [7] as a subroutine. The algorithm finds an approximately minimum-weight subgraph that has λ openly disjoint paths to a set of "root" vertices. This algorithm is used by all our vertex connectivity algorithms.

The input is an integer λ, a λ-vertex-connected undirected graph $G = (V, E)$, a nonnegative weight function w defined on the edges, and a set R of λ vertices. The output is an approximately minimum-weight subgraph of G in which there are λ openly disjoint paths from any vertex v to R. Observe that v could be in R, in which case we find $\lambda - 1$ paths from v to the remaining vertices in R.

The algorithm UNDIRECTED-FT(G, λ, R) is as follows. First create the directed version D of G in the most natural way: for each undirected edge in G create anti-parallel directed edges in D, each having the same weight as the corresponding undirected edge. Augment D by adding a new vertex r and add λ new directed edges of weight 0 from r to each vertex in R. Use the algorithm of Frank and Tardos on this graph with r as the root and find a minimum weight subgraph H with λ openly disjoint paths between r and any vertex of D. Let $S \subseteq E$ be those edges in G such that at least one of its copies is in H. Since S was obtained from H, for any vertex v in G, there are λ openly disjoint paths between v and R in S.

Proposition 4.1 *There are λ openly disjoint paths in S between any vertex in $G - R$ and R. There are $\lambda - 1$ openly disjoint paths in S between any vertex $v \in R$ and $R - \{v\}$.*

Lemma 4.2 *The weight of S is at most twice the weight of a minimum-weight λ-vertex-connected spanning subgraph of G.*

Proof. Let T be a minimum-weight λ-vertex-connected spanning subgraph of G. We observe that the directed version of T obtained as above by taking two anti-parallel directed edges instead of each undirected edge of T (with the same weight) is a subgraph of D, and along with the zero-weight edges from r to the vertices of R has λ openly disjoint paths from r to each vertex of G. Since the algorithm of Frank and Tardos returns a minimum-weight solution, the weight of the solution that it returns, $w(S)$, can be no heavier than $2w(T)$. ◻

We can also prove the following interesting theorem that S is at least $\lceil \lambda/2 \rceil$-vertex-connected. No assumptions about triangle inequality are required for this theorem. In other words, one can obtain a subgraph with half the required connectivity, paying at most twice the optimal cost (of twice the connectivity). A similar result (with a slightly better approximation ratio) can be derived from the work of Ravi and Williamson [21] by running their algorithm for $\lambda/2$ phases.

Theorem 4.3 *The vertex connectivity of S is at least $\lceil \lambda/2 \rceil$.*

Proof. We give a proof by contradiction. Assume that the graph contains a vertex cut C with $|C| < \lceil \lambda/2 \rceil$, i. e., the removal of C breaks G into components $C_1, C_2, \ldots, C_\ell$, where $\ell \geq 2$. Let C_i be a component such that $|C_i \cap R| \leq \lfloor \lambda/2 \rfloor$; clearly, such a component must exist. If C_i contains a vertex $v \notin R$, then there are at most $\lfloor \lambda/2 \rfloor + \lceil \lambda/2 \rceil - 1$ paths from v to vertices in R (the paths either must go to the vertices in $C_i \cap R$ or must go through the vertices in C). Clearly, this is a contradiction to the assumption that there were λ openly disjoint paths from v to R. If $C_i \cap R = C_i$, then there are at most $|C_i| - 1 + \lceil \lambda/2 \rceil - 1$ paths from vertices in R to any vertex in C_i. Since $C_i \leq \lfloor \lambda/2 \rfloor$, we conclude that there are at most $\lfloor \lambda/2 \rfloor + \lceil \lambda/2 \rceil - 2$ paths from R to any vertex in C_i. This is a contradiction to the assumption that there are $\lambda - 1$ openly disjoint paths from a vertex $v \in R$ to the remaining vertices in R. ◻

4.2 Weighted biconnectivity

We now describe a 2-approximation algorithm for weighted biconnectivity ($k = 2$).

Let $e = (x, y)$ be a minimum-weight edge in G. Set $R = \{x, y\}$ and call the procedure UNDIRECTED-FT$(G, 2, R)$. This returns a subgraph S of G. The biconnectivity algorithm now returns $S \cup \{e\}$ as the solution. Since S was obtained by calling UNDIRECTED-FT, by Proposition 4.1, for any vertex v in $G - \{x, y\}$, there are two openly disjoint paths starting at v, one ending at x and the other ending at y.

Lemma 4.4 *The graph $S \cup \{e\}$ is biconnected.*

Proof. We give a proof by contradiction. Suppose $S \cup \{e\}$ contains a cut vertex a. Let the deletion of a from S break the graph into components $C_1, \ldots, C_\ell$. Since x and y are adjacent they will be in $a \cup C_i$ (for some i). Consider a vertex $v \in C_j$ for some $j \neq i$. There cannot be two openly disjoint paths from v to x and y respectively, since both paths must go through a. But by Proposition 4.1 such paths must exist, which is a contradiction. ◻

Theorem 4.5 *The total weight of $S \cup \{e\}$ is at most $2 + 1/n$ times a minimum-weight biconnected spanning subgraph of G.*

Proof. Let $\mathcal{OPT}(G)$ be a minimum-weight biconnected spanning subgraph of G. Since any biconnected graph on n vertices contains at least n edges, a minimum weight edge of G is at most $w(\mathcal{OPT}(G))/n$. By Lemma 4.2, the weight of S is at most $2w(\mathcal{OPT}(G))$. □

4.3 Weighted k-vertex-connectivity

In this section we consider the weighted k-vertex-connectivity problem, when the edge weights satisfy triangle inequality. We describe an algorithm that finds a k-vertex-connected subgraph whose weight is within a factor $2+2(k-1)/n$ of a minimum-weight k-vertex-connected subgraph in G. Previously, an algorithm achieving a factor of 1.5 for the case $k = 2$ was given by Frederickson and JáJá [9].

Many algorithms for graphs satisfying triangle inequality are based on simple ideas for shortcutting. For example, taking two copies of a minimum spanning tree and then shortcutting it suitably yields a 2-approximation for 2-connectivity. One may wonder whether such a simple algorithm exists for the k-connectivity problem under triangle inequality. We first observe that the following straightforward algorithm does not yield a good approximation ratio. Take a TSP tour obtained as described above. Connect each vertex of the tour to the $k/2$ vertices that come before (and after) it on the tour. Such a graph is indeed k-connected, but there are instances in which its weight is $\Omega(k^2)$ times the weight of a minimum spanning tree and the performance ratio of the algorithm is $\Omega(k)$.

Our algorithm first finds a subset R of k vertices, such that the complete graph induced on R is relatively "light". It then calls Undirected-FT(G, k, R) to obtain a subgraph S of G. The algorithm returns $S \cup K_R$ as its output, where K_R is the complete subgraph on R.

We now show how to find the subset R of k vertices, such that the complete graph induced on the vertices in R is "light".

Let $S(i)$ be the "star" graph formed by vertex i together with the edges to its $k-1$ closest neighbors. Let j be a vertex with the weight of $S(j)$ being minimum over all vertices of G. Select R to be the vertices of $S(j)$, i. e., R consists of vertex j and its $k-1$ nearest neighbors. Let K_R be the complete graph on the vertices of R.

Lemma 4.6 *The weight of K_R, the complete graph on the vertices in R is at most $2(k-1)/n$ times the weight of $\mathcal{OPT}(G)$, a minimum weight k vertex-connected subgraph of G.*

Proof. Let $N(i)$ be the subgraph of $\mathcal{OPT}(G)$ formed by vertex i together with its $k-1$ closest neighbors in $\mathcal{OPT}(G)$ (observe that each vertex in $\mathcal{OPT}(G)$ has at least k neighbors). Clearly, for any i, $w(S(i)) \le w(N(i))$. Also since each edge is counted at most twice,

$$\sum_{i=1}^{n} w(N(i)) \le 2w(\mathcal{OPT}(G)).$$

Thus

$$w(S(j)) = \min_i w(S(i)) \le \frac{2}{n} w(\mathcal{OPT}(G)).$$

This is an upper bound on the weight of the star-graph centered at vertex j. For any pair of vertices $u, v \in S(j)$, by triangle inequality, we know that $w(u,v) \leq w(u,j) + w(j,v)$. Thus

$$w(K_R) = \sum_{u \in R} \sum_{v \in R, u<v} w(u,v) \leq \sum_{u \in R} \sum_{v \in R, u<v} w(u,j) + w(v,j) = (k-1)w(S(j)).$$

Thus the weight of K_R is at most $2(k-1)/n$ times $w(\mathcal{OPT}(G))$. □

Lemma 4.7 *The graph induced by the edges of $S \cup K_R$ is k-vertex-connected.*

Proof. We give a proof by contradiction. Assume that the graph contains a vertex cut C with $|C| < k$, i.e., the removal of C breaks G into components $C_1, C_2, \ldots, C_\ell$, where $\ell \geq 2$. Since any two vertices in R are adjacent, all vertices of R belong to $C \cup C_i$ (for some i). Consider a vertex $v \in C_j$ for some $j \neq i$. By Proposition 4.1 there must be k openly disjoint paths between v and R. But clearly there cannot be k openly disjoint paths from v to the nodes in R since these paths can only go through nodes in C, which is a contradiction. □

Theorem 4.8 *The total weight of $S \cup K_R$ is at most $(2 + 2(k-1)/n)w(\mathcal{OPT}(G))$.*

Proof. By Lemma 4.2, the weight of S is at most $2w(\mathcal{OPT}(G))$, and by Lemma 4.6, the weight of K_R is at most $2(k-1)/n \cdot w(\mathcal{OPT}(G))$. □

Remark: We note that K_R can be replaced by any subgraph S_R of G that has the property that there are k openly disjoint paths between every pair of vertices in R. If the weight of S_R is within a factor of α times $w(\mathcal{OPT}(g))$, then the above proofs can be modified to show that $S_R \cup S$ is k-connected and its weight is at most $\alpha + 2$ times $w(\mathcal{OPT}(g))$.

5 Conclusions

We conclude with some open problems related to the topic of obtaining approximation algorithms for the k-connected spanning subgraph problem.

1. Can we obtain an approximation factor better than 2 for the unweighted k-vertex-connected spanning subgraph problem? For the edge connectivity case we establish a 1.85 factor in this paper. Can this be improved further?

2. Can we obtain an approximation factor of 2 for the weighted k-vertex-connectivity problem even when the edge weights do not satisfy triangle inequality? For the edge connectivity case, a factor 2 approximation is known [19]. Can this be improved further?

Acknowledgements: We thank Randeep Bhatia, Hal Gabow, Kazuo Iwano, Michal Penn, Ramki Thurimella and Neal Young for useful discussions related to the problems and techniques discussed in this paper.

References

[1] M. Aggarwal, N. Garg, A scaling technique for better network design, *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 233–240, (1994).

[2] B. Bollobás, Extremal graph theory, *Academic Press, London*, (1978).

[3] J. Cheriyan, M.-Y. Kao and R. Thurimella, Algorithms for parallel k-vertex connectivity and sparse certificates, *SIAM J. Comput.*, 22 (1), pp. 157–174, (1993).

[4] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1975.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, *The MIT Press*, 1989.

[6] J. Edmonds, Matroid intersection, *Annals of Discrete Math.*, No. 4, pp. 185–204, (1979).

[7] A. Frank and E. Tardos, An application of submodular flows, *Linear Algebra and its Applications*, 114/115, pp. 320–348, (1989).

[8] G. N. Frederickson and J. JáJá, Approximation algorithms for several graph augmentation problems, *SIAM J. Comput.*, 10 (2), pp. 270–283, (1981).

[9] G. N. Frederickson and J. JáJá, On the relationship between the biconnectivity augmentation and traveling salesman problems, *Theoret. Comput. Sci.*, 19 (2), pp. 189–201, (1982).

[10] H. N. Gabow, A representation for crossing set families with applications to submodular flow problems, *Proc. 4th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 202–211, (1993).

[11] H. N. Gabow, M. X. Goemans and D. P. Williamson, An efficient approximation algorithm for the survivable network design problem, *Proc. 3rd Integer Programming and Combinatorial Optimization Conference*, pp. 57–74, (1993).

[12] M. R. Garey and D. S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, Freeman, San Francisco, 1979.

[13] N. Garg, V. Santosh and A. Singla, Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques, *Proc. 4th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 103–111, (1993).

[14] M. X. Goemans, A. V. Goldberg, S. Plotkin, D. Shmoys, E. Tardos and D. P. Williamson, Improved approximation algorithms for network design problems, *Proc. 5th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 223–232, (1994).

[15] M. X. Goemans and D. P. Williamson, A general approximation technique for constrained forest problems, *Proc. 3rd Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 307–316, (1992).

[16] M. Grötschel, C. Monma and M. Stoer, Design of survivable networks, *Handbook in Operations Research and Management Science*, Volume on Networks, 1993.

[17] D. Karger, Random sampling in cut, flow, and network design problems, *Proc. 26th Annual ACM Symposium on Theory of Computing*, pp. 648–657, (1994).

[18] S. Khuller and R. Thurimella, Approximation algorithms for graph augmentation, *J. Algorithms*, 14 (2), pp. 214–225, (1993).

[19] S. Khuller and U. Vishkin, Biconnectivity approximations and graph carvings, *J. Assoc. Comput. Mach.*, 41 (2), pp. 214–235, (1994).

[20] H. Nagamochi and T. Ibaraki, Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph, *Algorithmica*, 7 (5/6), pp. 583–596, (1992).

[21] R. Ravi and D. P. Williamson, An approximation algorithm for minimum-cost vertex-connectivity problems, *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 332–341, (1995).

[22] D. P. Williamson, M. X. Goemans, M. Mihail and V. V. Vazirani, A primal-dual approximation algorithm for generalized Steiner network problems, *Proc. 25th Annual ACM Symposium on Theory of Computing*, pp. 708–717, (1993).