Approximation Algorithms for Steiner Augmentations for Two-Connectivity

R. Ravi

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-92-21
April 1992

Approximation algorithms for Steiner augmentations for two-connectivity

R. Ravi*

Brown University

Abstract

We consider the problem of increasing the connectivity of a given graph to two at optimal cost. Fredrickson and Ja'Ja' [6] showed that that this problem is NP-hard and provided approximation algorithms. Recently, Khuller and Thurimella [7] have extended these results and presented a more efficient version of the results in [6]. We consider an extension of this problem to a more general setting. In this setting, we are given an edge-weighted graph, a subset of the vertex set called the sites and an initial subgraph; The objective is to find a minimum weight augmentation of edges that makes the sites two edge-connected or vertex-connected. We present approximation algorithms for the first time for these problems by extending those in [7] with matching performance bounds. If the initial subgraph connects all the sites, we find an augmentation of weight at most twice the optimal. Otherwise, the performance guarantee is three. The performance guarantee of three is obtained by an interesting application of an approximate min-max relation concerning Steiner trees and packing of certain kind of cuts derived in [1].

AMS 1980 Subject Classification: 68R10, 68Q25
Keywords: Approximation algorithms, augmentation, minimum-weight two-connected Steiner subgraphs, biconnectivity, bridge-connectivity, NP-complete problem.

1 Introduction

We consider the following two problems: Let $G = (V, E)$ be an undirected graph with a nonnegative weight function w on its edges. A subset S of vertices is specified as the *sites*. Given an initial subgraph $G_0 = (V_0, E_0)$ of G, we wish to find an edge-augmentation of minimum-weight such that there are two edge- (vertex-) disjoint paths between every pair of sites when we augment G_0 with the edges in the augmentation. We call these the Steiner bridge-connected and biconnected augmentation problems respectively. As the names suggest, these are generalizations of the problems of finding a minimum-weight bridge-connected and biconnected augmentations of a given edge-weighted graph (in the same way that the Steiner tree problem is a generalization of the minimum spanning tree problem). The latter problems were shown to be NP-complete by Fredrickson and Ja'Ja' [6] and they also offer approximation algorithms with a worst case performance guarantee of two when the initial subgraphs are connected. Their techniques also provide us with approximation algorithms for the minimum-weight bridge-connected and biconnected subgraphs of a given edge-weighted graph with performance guarantee three: start with a minimum weight spanning tree and augment it to achieve two connectivity.

In this note, we extend these techniques to the generalized problems. Also, we observe that a naive extension of the techniques of Fredrickson and Ja'Ja' to the minimum-weight two-connected Steiner subgraph problems would result in approximation algorithms with performance factor greater than three; The initial subgraph of minimum-weight that one-connects the sites is a minimum-weight Steiner tree which is NP-hard to compute exactly [5]. Hence, in this case, the performance guarantee is the sum of that for the Steiner tree problem and two for the second phase of augmenting this Steiner tree. This is

*Research supported by an IBM Graduate Fellowship. Additional support provided by NSF PYI award CCR-9157620 and DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

always greater than three. However, we adapt and use an approximate min-max equality relating Steiner trees to packing of cuts derived earlier [1] to improve the performance ratio to three.

Khuller and Thurimella [7] have recently presented a more efficient and concise version of the results in [6]. We follow their outline in our exposition here.

2 Definitions

We use the same terminology as in [7]. We are given an undirected graph $G = (V, E)$ with a nonnegative weight function w on its edges and $|V| = n$, $|E| = m$. The set E represents the set of feasible edges to be used in the subgraph. The initial subgraph to be augmented to be two-connected is denoted by $G_0 = (V_0, E_0)$. The weight of any subgraph G' is the sum of the weights of the edges in it. An undirected edge between two nodes u and v is represented by (u, v) and a directed edge from u to v by $\langle u, v \rangle$.

A directed graph is *strongly connected* if for every pair of vertices u and v in it, there is a directed path from u to v.

An undirected graph is *(one-)connected* if for any two vertices u and v, there is an undirected path from u to v. A *bridge* is an edge whose removal leaves the graph disconnected. A graph is *bridge-connected* if it contains no bridges. A *cut vertex* is a vertex whose removal along with its incident edges disconnects G. A graph is *biconnected* if it contains no cut vertices. Note that in a bridge-connected (biconnected) graph, there always exists two edge- (node-) disjoint paths between any pair of vertices.

A *block* is a maximal biconnected subgraph of a given undirected graph. Let $B^i = (V^i, E^i)$ be the blocks of an undirected graph G for $1 \leq i \leq t$ and let V_c represent the cut vertices in G. The *block-cut vertex tree* $T = (V_s, E_s)$ is defined as follows. Let V_b be a set of t new block vertices representing the blocks of G. Then $V_s = V_c \cup V_b$. Define $E_s = \{(c_i, B_j) \ : \ B_j \in V_b, c_i \in V_c \text{ and } c_i \in V^j\}$. Each edge connects a block vertex B_j with a cut vertex c_i that is in the block corresponding to B_j. Hence cut vertices and block vertices alternate in the block-cut vertex tree. Associated with each vertex in the block-cut vertex tree is a set of nodes of the graph. If c_i is a cut vertex, then the set corresponding to the cut vertex $c_i \in V_c$ is just $X_i = \{c_i\}$. If $B_j \in V_b$, the set associated with it is $Y_j = \{v \ : \ v \in V^j \text{ and } v \text{ is not a cut vertex }\}$. If G is a tree, then each block corresponds to a single edge in the tree and all the vertices except the leaves of the tree are cut vertices. From these definitions, we have the following observations.

Observation 2.1 *In the block-cut vertex tree T, each edge is between a vertex in V_b and a vertex in V_c.*

Observation 2.2 *The sets associated with the vertices of T form a partition of the vertices of G.*

An *arborescence* rooted at a vertex r is a directed spanning tree with all nodes except r having indegree one and r having indegree zero. A *reverse arborescence* rooted at r is one with edges directed towards r, i.e., all nodes except r have outdegree one and r has outdegree zero. Given a directed graph with edge-weights and a root node r, the algorithms in [2, 3, 9] can be used to find a minimum-weight arborescence.

For a vertex v in a rooted tree T, let the components formed by deleting v be denoted by $C_1(v), \ldots, C_{d(v)}(v)$ where $d(v)$ is the degree of v in T. If v is not the root, we will assume that $C_1(v)$ is the component containing the root and the other components correspond to the subtrees rooted at the children of v. In a rooted tree, for a vertex v, we denote its parent by $p(v)$.

3 Background

First, we introduce a few results that we shall be using later in the analysis of the performance guarantee of our algorithms.

3.1 Packing of requirement cuts

Given a graph G and a subset of the vertices S specified as sites, we define a *requirement cut* as a set of edges whose removal separates at least one site from another. Each requirement cut (henceforth referred to as a *cut*) can be written as $\Gamma(W)$, where W is a node subset of the graph, and $\Gamma(W)$ denotes the set

of edges with exactly one endpoint in W. $\Gamma(W)$ is also referred to as the *cocycle* determined by W. A fractional packing of cuts is a family of cuts $\Gamma(W_1), \Gamma(W_2), \ldots, \Gamma(W_k)$, together with a rational *weight* λ_i for each cut. A (fractional) *w-packing* of cuts is a weighted collection of cuts that have the following property: for each edge (u, v), the sum of the weights of all the cuts in this collection containing the edge is at most $w(u, v)$. The *value of the packing* is the sum of the weights of all the cuts in the packing. A *maximum packing* is one of maximum value.

3.2 Maximum packing of cuts and minimum-weight Steiner trees

In this subsection, connectivity refers to *both* edge and vertex connectivity unless explicitly stated otherwise. Given weights on the edges, the problem of finding a minimum-weight Steiner tree connecting all the sites is NP-complete. Since a requirement cut separates the sites, any Steiner tree must have an edge in the cut. Thus it is easy to see that the weight of any Steiner tree is at least as much as the value of a maximum packing of requirement cuts. The following result was shown in [1].

Theorem 3.1 *Given an undirected graph with edge-weights, and a subset of the vertices specified as sites, the minimum-weight Steiner tree connecting all the sites has weight at most twice the value of the maximum packing of requirement cuts.*

The algorithm in [1] greedily finds a packing of requirement cuts and simultaneously builds a Steiner tree of weight at most twice the value of this packing.

Note that any Steiner bridge-connected subgraph must have at least *two* edges crossing any requirement cut since this subgraph has two edge-disjoint connections between the separated sites. In fact, the same holds for any Steiner biconnected subgraph also since node-disjoint paths are also edge-disjoint. Thus we have the following lemma.

Lemma 3.1 *The weight of any two connected Steiner subgraph is at least twice as much as the value of a maximum packing of requirement cuts.*

Combining the results from the Theorem and the lemma above, we get the following theorem.

Theorem 3.2 *Given an undirected graph with edge-weights, and a subset of the vertices specified as sites, the approximately minimum-weight Steiner tree found by the algorithm in [1] has weight at most that of any minimum-weight two-connected Steiner subgraph.*

We can extend the above theorem to handle the case when all sites are not one-connected in the initial subgraph G_0. To one-connect the sites in initial subgraph, we do the following. We find the connected components of the initial subgraph. We mark those that contain a separated site as *significant*. Now, we contract the significant components to single supernodes; the uncontracted edges of G_0 are assigned zero weight. Now we find a Steiner tree of approximately minimum weight connecting these significant supernodes and the remaining sites in G_0 using the algorithm in [1]. This algorithm also gives us a packing of requirement cuts of at least half the value as the weight of the Steiner tree. This Steiner tree is our augmentation that one-connect the sites. Each requirement cut in the packing is also a requirement cut in G since it separates some sites. Also, the optimal augmentation for two-connectivity also must intersect each of these requirement cuts in the packing twice. Thus the cost of the optimal augmentation is at least twice the value of this cut packing. Putting these together, we get the following theorem.

Theorem 3.3 *Given an undirected graph with edge-weights, a subset of the vertices specified as sites and an initial subgraph which does not one-connect the sites, we can find an augmentation of the initial subgraph that one-connects the sites whose weight is at most that of an optimal augmentation that two-connects the sites.*

Thus in the case that all the sites are not one-connected in the initial subgraph, we first find an augmentation that one-connects all the sites as outlined above. By the theorem above, the weight of this augmentation is at most the weight of the optimal augmentation for two-connectivity. Then we use the algorithms for connected initial subgraphs detailed in later sections to obtain an augmentation of this graph with weight at most twice the optimal. Thus we have an approximation algorithm with guarantee three.

4 The Steiner bridge-connected augmentation problem

4.1 Overview

We follow the outline of the algorithm of Fredrickson and Ja'Ja' [6] as modified by Khuller and Thurimella [7] for finding an approximately minimum-weight bridge-connected augmentation of a given initial subgraph. First, note that we can assume that we are given an initial subgraph one-connecting all the sites, from the remarks in the end of the previous section. Notice then that it is sufficient to show how to augment a tree for bridge-connecting it. For, if the connected initial subgraph G_0 has non-trivial bridge-connected components, then we can shrink each of these components to single vertices, and form a tree whose edges correspond to the bridges in G_0. The edges retained in E are those of minimum weight that connect vertices of different bridge-connected components among themselves or to other nodes in G. We retain only the minimum weight edge in these cases since we are looking only for a minimum weight augmentation. Further, we can disregard the edges in E within the same bridge-connected component since they are of no use in augmenting G_0 to be bridge-connected. Thus we have modified the initial subgraph to a tree T_0.

Call a bridge-connected component in G_0 *significant* if it contains a separated site. Since we assumed that G_0 one-connects all the sites, each site is in some significant component in G_0. Thus each site is represented in some significant node in T_0. We further modify the tree T_0 so that all its leaves represent significant components. We contract each edge in T_0 whose removal from T_0 does not separate any pair of significant components. Let the resulting tree be denoted by T_0'. After contracting such edges, each node in T_0' represents a set of nodes in G. Moreover, as a result of the contractions, each leaf of T_0' represents a set of nodes containing at least one separated site. Any set of edges in $E - E_0$ whose addition to E_0 bridge-connects the sites also bridge connects all the leaves in T_0' and vice-versa. Thus we can assume without loss of generality that the input graph G_0 is a tree where each leaf of the tree is a site. We shall work under this assumption in the presentation of the algorithm.

Let V' denote the set of nodes in V connected by the edges in E_0. To begin, we form a multigraph G' only on the vertices V'. Namely $G' = (V', E' \cup E_0)$ where the edge $(u, v) \in E'$ whenever there is a path from u to v in G not using any edge in E_0 and $\cup$ represents multiset union. For $(u, v) \in E'$, we set $w(u, v) =$ the minimum weight of such a path under the w function. We say that the edge (u, v) so introduced in E' *corresponds* to a minimum-weight path between u and v in G not using any edge in E_0. From G', we construct a directed graph G^D, and find a minimum weight arborescence rooted at a vertex r. If there is no such arborescence, we can show that there is no augmentation possible using feasible edges. We use the edges in the arborescence to choose the augmentation solution and show that the cost is at most twice the optimal.

4.2 The algorithm

The algorithm takes as input the graph G' and the tree G_0. The algorithm outputs a set of edges $Aug \subseteq E - E_0$ that bridge-connects G_0.

(1) (*Construct* $G^D = (V', E_D)$)

(a) Pick a leaf r in G_0 and direct the edges in G_0 to form a reverse arborescence rooted at r (with all edges directed towards r). Denote the resulting tree by T.

(b) Add to E_D the directed tree edges of T and set their weight to zero.

(c) For each edge $(u, v) \in E'$, if (u, v) is a back edge (i.e. connects a vertex to one of its ancestors), we add one directed edge to E_D; otherwise, we add two directed edges to E_D as detailed below. We will call these directed edges as *images* of (u, v) and we say these directed edges are *generated* by (u, v). Let $e = (u, v) \in E'$ and have weight w (See Figure 1).

(i) If u is an ancestor of v (the converse is symmetric), then add an edge $\langle u, v \rangle$ in G^D with weight w. Note that u may be the parent of v.

(ii) If neither u nor v is an ancestor of the other, let $t = l.c.a.(u, v)$ be the least common ancestor of u and v in the tree T. Add the directed edges $\langle t, u \rangle$ and $\langle t, v \rangle$ in G^D each with weight w.

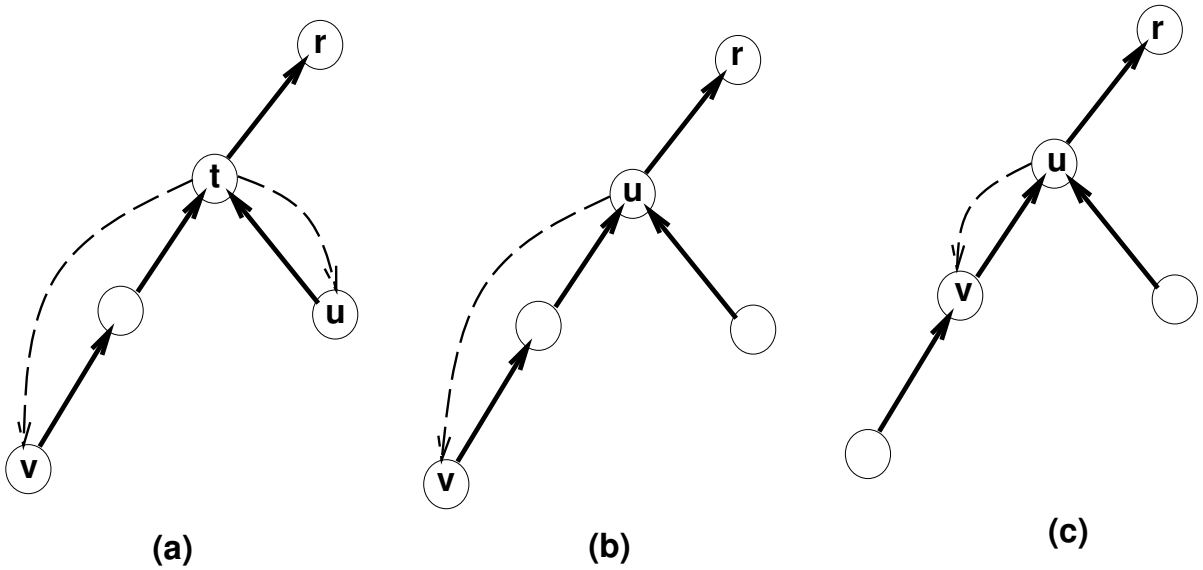

Figure 1: Cases in construction of G^D. In (a), neither u nor v is an ancestor of the other. In the other two cases, u is an ancestor of v. The node u may even be the parent of v as in (b) if there is a path in G not using edges in G_0 between u and v.

(2) Find a minimum weight arborescence rooted at r. Define $Aug = \{$ paths in G corresponding to (u, v) : a directed edge generated by (u, v) is chosen in the arborescence.$\}$

We have the following simple observation about the edges added in G^D.

Observation 4.1 *All the edges in $G^D - T$ connect a vertex to one of its descendants in T.*

4.3 Correctness and Guarantee

We now state and sketch the proofs of the key lemmas. They follow exactly the same outline as in [7].

Lemma 4.1 *If the set of sites is bridge-connected in G, then G^D is strongly connected.*

Proof Sketch: The proof is by contradiction. Clearly every vertex in G^D can reach the root r via the tree T. So if G^D is not strongly connected, there is some vertex that the root cannot reach. Consider such a vertex closest to the root and call it u. Now, it is easy to see any two sites are bridge connected in G' whenever they are bridge-connected in G. By our assumption about G_0, the root r, being a leaf, is a site and so is any leaf l that is a descendant of u. Moreover, r and l are bridge-connected in G and hence in G'. Thus the edge $(u, p(u))$ is not a bridge in G'. Since $(u, p(u))$ is not a bridge, we can find an edge (v, s) in G' between v, a descendant of u and s a non-descendant of u. This undirected edge causes addition of directed edges that make u reachable from r giving a contradiction. □

Lemma 4.2 *If G bridge-connects all the sites, then the graph G_0 along with Aug bridge-connects all the sites.*

Proof Sketch: By the previous lemma, we can find a minimum weight arborescence in G^D. Assume that G_0 along with Aug fails to bridge-connect all the sites. Then we can find a bridge edge $(u, p(u))$ in the tree T that is closest to the root (not necessarily unique). Since the descendants of u are reached from r by the arborescence, there must be a directed edge $\langle w, v \rangle$ from an ancestor, w, of u to a descendant, v, of u (using Observation 4.1). This edge must have been generated by an edge in G' connecting v to a node outside the subtree rooted at u which contradicts the assumption that $(u, p(u))$ is a bridge. □

Lemma 4.3 *The weight of Aug is at most twice that of the optimal augmentation of G_0 to bridge-connect the sites.*

Proof Sketch: We prove the lemma by exhibiting an arborescence of weight at most $2w(Aug^*)$, where Aug^* is the optimal augmentation. Define

$$E'' = \{(x,y) \ : \ x,y \in G_0 \text{ and there is a path not using any nodes in } G_0 \text{ between } x \text{ and } y \text{ in } Aug^* \ \}$$

The set of directed edges generated by the edges in E'' along with the zero weight edges in G_0 forms a strongly connected graph with total weight at most $2w(Aug^*)$. Since we find a minimum weight arborescence, the weight of the one we find is at most $2w(Aug^*)$ as well. □

The above lemmas prove the following theorem.

Theorem 4.1 *There is an approximation algorithm to find an augmentation of minimum-weight that bridge-connects a given set of sites of an edge-weighted undirected graph starting with an initial subgraph that one-connects the sites. The performance guarantee is two.*

As a corollary, combining remarks from the end of Section 3 summarized in Theorem 3.3, we get the following.

Corollary 4.1 *There is an approximation algorithm to find an augmentation of minimum-weight that bridge-connects a given set of sites of an edge-weighted undirected graph starting with an initial subgraph that does not connect the sites. The performance guarantee is three.*

As a special instance of this corollary, we can obtain approximation algorithms for the minimum weight subgraph that bridge-connects the sites with performance guarantee three. The costliest step in our algorithm is the computation of the all-pair shortest paths for determining the edges in E' and their weights which takes $O(n^3)$ time. We note that the above performance ratios are approachable as illustrated by an example in [6] for the case of minimum-weight spanning bridge-connected subgraphs. By using the same example with the set of sites being the whole node set of the graph, we see that our algorithm reduces to that in [6] and hence performs just as badly on this example.

5 The Steiner biconnected subgraph problem

5.1 Overview

Again, as in Section 4, we can assume without loss of generality that the initial subgraph G_0 connects the sites. First, we form a graph G' on the nodes of G_0 with edges of G' corresponding to paths that are node-disjoint from G_0. We then contract the blocks in G' to form a tree structure T. Using the graphs G' and T, we construct a weighted directed graph G^D. We then find a minimum-weight arborescence in G^D from which we identify the set of edges to be included in the augmentation. One important difference is that in this case, the tree T we use is the block-cut vertex tree of G_0.

As described in Section 4, we can modify the tree T to ensure that each leaf of the tree represents a set of nodes that contains a separated site. Thus we can assume without loss of generality that this is the case. In this case, it is easy to see that any augmentation of G_0 that biconnects the sites biconnects the leaves of T and vice-versa.

5.2 The algorithm

Let V' denote the set of nodes in V connected by the edges in E_0. To begin, we form a graph G' only on the vertices V_0. Namely $G' = (V', E' \cup E_0)$ where the edge $(u,v) \in E'$ whenever there is a path from u to v in G not using any nodes in E_0 and $\cup$ represents multiset union. Also, we set $w(u,v) =$ the weight of such a path of minimum weight under the w function. We say that the edge (u,v) so introduced in E' *corresponds* to such a path in G. We then construct the block-cut vertex tree T of G'. Using Observation 2.2, we can associate a vertex of the tree T meaningfully with each vertex of G'. By *superimposing* an edge (x,y) in G' on the tree T, we mean adding an edge between vertices a and b of T where their associated sets X and Y contain x and y respectively.

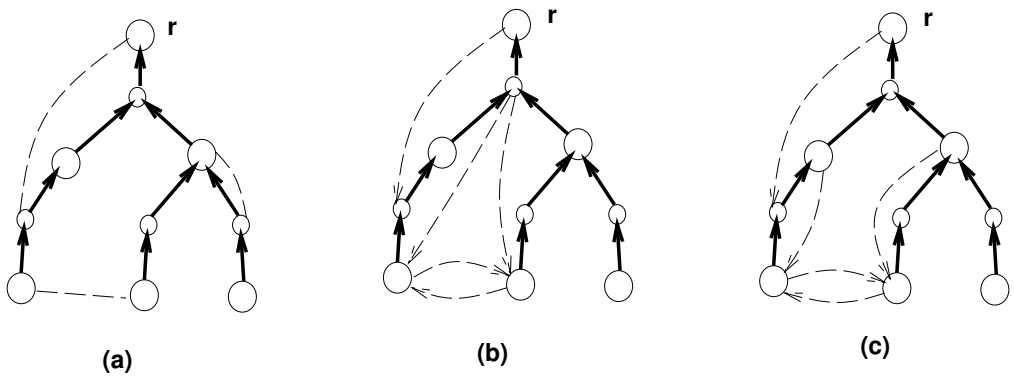

Figure 2: Stages in the construction of G^D. In (a), the dotted edges are the superimposed edges of E'. Alternating bigger and smaller nodes represent blocks and cut vertices in T. In (b), the graph G^D constructed after Step 2(c) is shown, and in (c), the graph is shown after performing the modifications described in Step 2(d).

The algorithm takes as input the graph G' and the block-cut vertex tree T of G'. The algorithm outputs a set of edges $Aug \subseteq E - E_0$ that bridge-connects G_0.

(1) Superimpose all the edges of E' on T. Discard all the self-loops. Among multiple parallel edges, retain one of minimum weight and discard the rest.

(2) ($Construct\ G^D = (V', E_D)$)

(a) Pick a leaf r in T and direct the edges in T to form a reverse arborescence rooted at r. Continue to denote this directed tree by T.

(b) Add to E_D the directed tree edges of T and set their weight to zero.

(c) Consider the superimposed edges of E' on T. Let (u, v) be such an edge. If (u, v) is a back edge, we add one directed edge to E_D; otherwise, we add four directed edges to E_D as detailed below. We call these directed edges as *images* of (u, v) and we say these directed edges are *generated* by (u, v). Let $e = (u, v) \in E'$ and have weight w (See Figure 2).

(i) If u is an ancestor of v (the converse is symmetric), then add an edge $\langle u, v\rangle$ in G^D with weight w. In the case that u is the parent of v, we discard this edge and add nothing to E_D (Note that such edges can be removed from *any* biconnectivity augmentation of G_0).

(ii) If neither u nor v is an ancestor of the other, let $t = l.c.a.(u, v)$ be the least common ancestor of u and v in the tree T. Add the four directed edges $\langle t, u\rangle, \langle t, v\rangle, \langle u, v\rangle$ and $\langle v, u\rangle$ in G^D each with weight w.

(d) Modify E_d as follows. For every $u \in V_c$, if there is an outgoing edge from u to a descendant v, then replace that edge with $\langle u_v, v\rangle$ where u_v is the child of u on the tree path from u to v.

(3) Find a minimum weight arborescence rooted at r. Define $Aug = \{$ path in G corresponding to (u, v) : a directed edge generated by (u, v) is chosen in the arborescence.$\}$

From the construction of G^D, we have the following observation.

Observation 5.1 *In the directed graph G^D, there are no outgoing edges from a cut vertex to any of its descendants in T.*

5.3 Correctness and Performance guarantee

We now state and sketch the proofs of the key lemmas. Again, we follow exactly the same outline as in [7].

Lemma 5.1 *If the set of sites is biconnected in G, then G^D is strongly connected.*

Proof Sketch: The proof is by contradiction. Clearly, every vertex in G^D can reach the root r via the tree T. So if G^D is not strongly connected, there is some vertex that the root cannot reach. Consider such a vertex closest to the root and call it u. There are two cases to consider. Let the vertex u be a cut vertex. The key observation is that the components formed on the deletion of vertex u from T are connected by the set of superimposed edges of E'. This is because the sites that form the leaves of T are biconnected in G. Using these connections, we can identify directed edges added to G^D that make u reachable from r giving a contradiction. The case when u is a block vertex is handled by a similar argument on the cut vertex $p(u)$. The details are in [7]. □

Lemma 5.2 *If G biconnects all the sites, then the graph G_0 along with Aug biconnects all the sites.*

Proof Sketch: By the previous lemma, we can find a minimum weight arborescence in G^D. Assume that G_0 along with Aug fails to biconnect all the sites. Then we can find a cut vertex u in the tree T. We then consider the components $C_1(u), \ldots, C_{d(u)}(u)$ and group them into two groups: the first containing C_1 and all the other components that are connected to C_1 when the edges of Aug are superimposed on T, and the second containing the rest. Since u is a cut vertex, both the groups must be non-empty. Furthermore, all the vertices in G^D are reachable from the root r in the minimum weight arborescence we find. Also, by Observation 5.1, no directed edges go out of u in G^D. These imply that there must be a directed edge from a component in the first group to one in the second. The path generated by this directed edge in Aug would contradict the assumption that u is a cut vertex. □

Lemma 5.3 *The weight of Aug is at most twice that of the optimal augmentation of G_0 to biconnect the sites.*

Proof Sketch: We prove the lemma by exhibiting an arborescence of weight at most $2w(Aug^*)$, where Aug^* is the optimal augmentation. Define

$$E'' = \{(x, y) \ : \ x, y \in G_0 \text{ and there is a nontree path not using any nodes in } G_0 \text{ between } x \text{ and } y \text{ in } Aug^* \ \}$$

The set of directed edges generated by the edges in E'' along with the zero weight edges in G_0 forms a strongly connected graph with total weight at most $4w(Aug^*)$. However, no arborescence will use more than two of the four directed images of each edge in E''. Since we find a minimum-weight arborescence, the weight of the one we find is at most $2w(Aug^*)$. □

The above lemmas prove the following theorem.

Theorem 5.1 *There is an approximation algorithm to find an augmentation that biconnects a given set of sites of an edge-weighted undirected graph starting with an initial subgraph that one-connects the sites. The performance guarantee is two.*

As a corollary, using Theorem 3.3, we get the following.

Corollary 5.1 *There is an approximation algorithm to find an augmentation that biconnects a given set of sites of an edge-weighted undirected graph starting with an initial subgraph that does not connect the sites. The performance guarantee is three.*

As a special instance of this corollary, we can obtain approximation algorithms for the minimum weight subgraph that biconnects the sites with performance guarantee three. Again, the costliest step is the computation of the all-pair shortest paths for determining the edges in E' and their weights which takes $O(n^3)$ time.

6 Conclusions

Though more recent algorithms in [8] solve the minimum bridge-connected and biconnected subgraph problems with better performance factors, they do not extend to the edge-weighted or to the Steiner cases we considered. While the method described in [6] generalizes to the Steiner setting as we saw above, it does not extend to the generalized Steiner setting where we seek to two-connect arbitrarily specified pairs of vertices. This remains an important open problem.

Acknowledgements

It is a pleasure to thank my advisor, Philip Klein, for his encouragement and help in verifying these results.

References

[1] A. Agrawal, P. Klein and R. Ravi, "When trees collide: an approximation algorithm for the generalized Steiner problem on networks," *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (1991), pp. 134-144.

[2] Y. J. Chu, and T. H. Liu, "On the shortest arborescence of a directed graph", *Sci. Sinica, 14* (1965), pp. 1396-1400.

[3] J. Edmonds, "Optimum branchings", *J. Res. Nat. Bur. Standards Sect. B, 71* (1967), pp. 233-240.

[4] K. P. Eswaran, and R. E. Tarjan, "Augmentation problems", *SIAM Journal on Computing, Vol. 5* (1976), pp. 653-665.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco (1979).

[6] Greg N. Fredrickson, and Joseph Ja'Ja', "Approximation algorithms for several graph augmentation problems", *SIAM Journal on Computing, Vol. 10, No. 2* (1981), pp. 270-283.

[7] S. Khuller, and R. Thurimella, "Approximation algorithms for graph augmentation," Technical report UMIACS-TR-91-132, CS-TR-2766, Univ. of Maryland, September (1991), to appear in *Journal of Algorithms.*

[8] S. Khuller, and U. Vishkin, "Biconnectivity approximations and graph carvings", Technical Report UMIACS-TR-92-5, CS-TR-2825 Univ. of Maryland, January 1992.

[9] R. E. Tarjan, "Finding optimum branchings", *Networks, 7* (1977), pp. 25-35.