Approximation Algorithms for Graph Augmentation

Samir Khuller [*]   *Ramakrishna Thurimella* [†]

Abstract

We study the problem of increasing the connectivity[1] of a graph at an optimal cost. Since the general problem is NP-hard, we focus on efficient approximation schemes that come within a constant factor from the optimal. Previous algorithms either do not take edge costs into consideration, or run slower than our algorithm. Our algorithm takes as input an undirected graph $G_0 = (V, E_0)$ on n vertices, that is not necessarily connected, and a set *Feasible* of m weighted edges on V, and outputs a subset Aug of edges which when added to G_0 make it 2-connected. The weight of Aug, when G_0 is initially connected, is no more than twice the weight of the least weight subset of edges of *Feasible* that increases the connectivity to 2. The running time of our algorithm is $O(m + n \log n)$. We also study the problem of increasing the edge connectivity of any graph G, to k, within a factor of 2 (for any $k > 0$). The running time of this algorithm is $O(nk \log n(m + n \log n))$. We observe that when k is *odd* we can use different techniques to obtain an approximation factor of 2 for increasing edge connectivity from k to $(k + 1)$ in $O(kn^2)$ time.

1 Introduction

Augmenting the connectivity of communication networks is increasingly becoming important to provide reliable means of communication. Informally, the problem is the following: we have a "current" network, and we wish to increase the connectivity of the network by adding edges. Each *feasible* edge has an associated weight, and we want to achieve the desired connectivity by adding the least total weight of edges.

The Weighted Augmentation Problem: Consider a graph $G = (V, E)$ with a weight $w_i \geq 0$ on edge e_i, where E is the set of feasible edges. We are also given a "current" network, the graph $G_0(V, E_0)$ that is a subgraph of G. The goal is to add a minimum weight set of edges Aug, to G_0, such that the resulting graph is k-connected for a given k. The edges that we are permitted to add, are required to be edges from the graph G (the feasibility graph). If $E_0 = \Phi$ and $k = 1$ then the problem is that of finding a minimum weight spanning tree on V. It is well known that this problem can be solved very efficiently. For $k > 1$, the problem turns out to be NP-hard. We will study approximation algorithms for the general problem.

Our Results: Let m and n denote the number of edges and the number of vertices, respectively, of the input graph. In this paper we provide a simple algorithm (for $k = 2$) that produces an approximation within a factor of 2 in $O(m + n \log n)$ time. Note that for sparse networks the algorithm is faster by a factor of n^2/m. (The previous algorithm for the problem is due to

[*]Institute for Advanced Computer Studies (UMIACS), The University of Maryland at College Park, College Park, MD 20742. This work is partially supported by NSF grants CCR-8906949, CCR-9103135 and CCR-9111348.

[†]Department of Mathematics and Computer Science, University of Denver, Denver, CO 80208. Part of this work was done while this author was with UMIACS.

[1]Connectivity refers to both edge and vertex connectivity throughout the paper unless stated explicitly otherwise.

[FJ81] and takes $O(n^2)$ time.) We design algorithms for both edge and vertex connectivity. The speedup is obtained not only by using a better subroutine for branchings in directed graphs, but by also doing away with the dynamic programming subroutine (computation of function $DIST$) used by [FJ81]. Since the subroutine $DIST$ used by [FJ81] computes $\Theta(n^2)$ values there is no way to make their algorithm more efficient. It should be noted however that our strategy is similar to that of [FJ81].

We also show that an approximation factor of 2 can be obtained to augment the edge connectivity of any graph to being k-edge connected. The running time of this algorithm is $O(nk \log n(m + n \log n))$. This improves the approximation factor of 3 that was given by [FJ81] to increase the edge connectivity of an unconnected graph to 2, albeit the running time of our scheme is significantly higher. When k is *odd* we show how to obtain the same approximation factor for increasing edge connectivity from k to $(k + 1)$ in $O(kn^2)$ time.

Significance of the Results: The subtlety of the construction, and the manner in which it is combined with the branching gives the scheme its simplicity and elegance.

The application of algorithms for finding k-edge disjoint branchings (for directed graphs) to incrementing edge connectivity in undirected graphs appears to have gone unnoticed until now. This gives the first polynomial time algorithm for obtaining the approximation factor of 2 for incrementing the edge connectivity of *any* graph to *any* k (when the edges have weights). The k-edge disjoint branching problem is solved by formulating it as a weighted matroid intersection problem [Ed79, G91a]. It remains of interest to see how the solutions to various problems in the area of Matroids will give rise to approximation schemes for NP-hard problems.

Previous Work: The problem of increasing the connectivity to 2 was first considered by [ET76], who showed that it was NP-hard. In subsequent work, Frederickson and JáJá [FJ81] showed that the problem remained NP-hard even if G_0 is connected. They also provided approximation algorithms to solve the problem with a running time of $O(n^2)$ and produced an approximation within a factor of 2 of the optimal solution [FJ81] when G_0 is initially connected. For the special case of the edge weights satisfying the triangle inequality they designed algorithms that produce an approximation within a factor of $\frac{3}{2}$ of the optimal solution [FJ82].

The more general problem of increasing the edge connectivity for *any* k has not been studied in the weighted case to our knowledge. We also observe that the NP-hardness proof of [FJ81] extends to work for any $k > 0$.

Related Work: When the feasible set of edges is the complete graph, and all the edges have unit weight, the augmentation can be done optimally to obtain any desired edge connectivity in polynomial time. In this setting, there have been a number of papers that have studied graph augmentation problems, starting from the seminal paper by Eswaran and Tarjan who first introduced the problem of making a connected graph biconnected by adding the smallest set of edges [ET76]. A linear time implementation was provided by Rosenthal and Goldner [RG77] (see also [HR91b]). This has recently been extended to triconnecting a graph optimally by Hsu and Ramachandran [HR91a]. Subsequent work has led to the generalization of the problem in many ways, and also led to faster algorithms. This was first shown by Watanabe and Nakamura [WN87, Wa88]. Recently much faster algorithms have been obtained by Naor, Gusfield and Martel [NGM90] and by Gabow [G91b]. A generalization of the problem was considered by Frank [Fr90] who has shown a more general problem to be solvable in polynomial time, but the algorithm is not very efficient. In the context of planar graphs, [KB91] have given approximation algorithms for the case when edges have to be added, maintaining planarity.

For weighted directed graphs the following problem is solvable in polynomial time [Ed79, FT89, G91a]. In a digraph find a minimum weighted set of edges so as to have k-edge disjoint paths from a source to every vertex. Gabow provides the fastest implementation of Edmond's algorithm [G91a]. We use this algorithm to obtain an approximation factor of 2 for the edge connectivity augmentation problem in undirected graphs.

2 Preliminaries

A graph is said to be k-vertex (k-edge) connected if it has at least $(k+1)$ vertices (edges), and the deletion of any $(k-1)$ vertices (edges) leaves the graph connected. A *branching* of a directed graph G rooted at a vertex r is a spanning tree of G such that each vertex except r has indegree exactly 1 and r has indegree zero. The *weight* of a branching is the sum of the weights of the edges of the branching. A *minimum* weight branching is a branching with the least weight. For the definitions of bridges, 2-edge (2-vertex) connected components the reader is referred to [Ev79]. The 2-vertex connected components of a graph are also referred to as *blocks*. For a vertex v in a rooted tree Γ, let the components formed by the deletion of v be called $C_1(v), C_2(v), \ldots, C_{d(v)}(v)$, where $d(v)$ is the degree of v in Γ. If v is not the root, we will assume that $C_1(v)$ is the component that contains the root, and the other components correspond to subtrees rooted at the children of vertex v. In a rooted tree, for a vertex u we denote its parent by $p(u)$.

As a matter of notation, we refer to an undirected edge between two vertices x and y as (x, y). On the other hand, a directed edge from x to y is denoted by $x \rightarrow y$.

We can assume that G_0 is a connected graph. If the graph is not initially connected, find a minimum spanning tree and add the edges of the tree to G_0. This would yield an approximation within a factor of 3 to the problem of finding a least-weight 2-connected spanning subgraph of a 2-connected weighted graph. We can also assume that G is 2-connected.

3 Increasing Edge Connectivity from 1 to 2

Notice that we only need to show how to increase the edge connectivity of a tree due to the following observation. If we are given G_0 with nontrivial 2-edge connected components, then we can shrink the vertex sets of these components into single vertices, resulting in a tree whose edges are the bridges of G_0. The edges to be retained from *Feasible* are the minimum weight edges that connect vertices of different 2-edge connected components of G_0. (Observe that the edges of *Feasible* that connect vertices of the same 2-edge connected component are of no use in augmenting G_0. Similarly, among the edges that connect different 2-edge connected components only the minimum weight edge is of interest.)

From G_0, we will construct a directed graph G^D, and find a minimum weight branching from a vertex r. In case there is no branching that spans all the vertices, we can show that there is no way to increase the connectivity of the network. Using a minimum weight branching of G^D, we can find a set of edges of $G - G_0$ whose addition will increase the connectivity of G_0. We can also show that the total weight of the edges added by this technique is bounded by twice the weight of an optimal augmentation.

Algorithm *Find Edge Aug*
Input: Graph G and a spanning tree G_0.
Output: A set of edges $Aug \subseteq E - E_0$.

```
(1) (Construct $G^D = (V, E_D)$)
    (a) Pick an arbitrary leaf $r$ and root the tree $G_0$ at $r$ by directing all the edges towards
        the root. Denote the resulting tree by $\Gamma$.
    (b) Add to $E_D$ the directed tree edges of $\Gamma$ and set their weight to zero.
```

(c) Consider the edges that belong to $G = (V, E)$ but do not belong to G_0 (edges in $E - E_0$). For each such edge (u, v), if (u, v) is a back edge (i.e., it connects a vertex to one of its ancestors), we add one directed edge to E^D (shown below); otherwise, we add two directed edges to E^D. (We will refer to these directed edges as *images* of (u, v), and we say these directed edges are *generated* by (u, v).)

Suppose that the edge e with weight w, joins vertices u and v belonging to the tree Γ. There are two cases depending on the relative locations of u and v in the tree Γ. (See Figure 1.)

(i) If u is an ancestor of v (the converse is symmetric): then add an edge $u \rightarrow v$ in G^D with weight w.

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

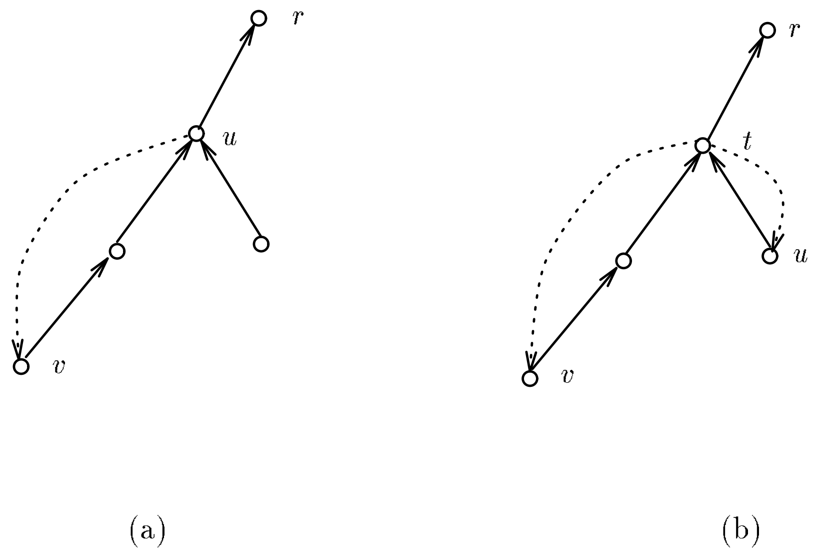


Figure 1: Construction of G^D.

(2) Find a minimum weight branching in G^D rooted at r. For each directed edge e that is picked as part of the branching, and that does not belong to the directed tree Γ, add the corresponding edge in $E - E_0$ that generated e. The set of edges added is Aug.

Observe that all edges of $G^D - \Gamma$ are such that they connect a vertex to one of its descendants (in Γ). We now show the following.

Lemma 3.1 *If G is 2-edge connected, then the directed graph G^D is strongly connected.*

Proof: Clearly, all the vertices of G^D can reach the root r using edges from the tree Γ. Further, let us assume that G^D is not strongly connected. Of all the vertices that cannot be reached from the root, let u be the vertex that is closest to the root in Γ. Clearly, the entire subtree rooted at u must consist of unreachable vertices. Since the image of the edge $(u, p(u))$ in G is not a bridge in G, there must be another edge (v, s) in G going from a vertex v that is in the subtree rooted at u, to vertex s that is not in this subtree. Such an edge would have generated a directed edge from a vertex w to v in G^D where w is an ancestor of v (specifically the least-common-ancestor of v and s). Since w is a proper ancestor of u, it is reachable from r in G^D. Therefore v is reachable from r, and hence u as well. Thus we obtain a contradiction. □

Lemma 3.2 *If G is 2-edge connected, then the edge connectivity of the graph G_0 together with the edges in Aug is at least 2.*

Proof: Assume G is 2-edge connected. Then, by the previous lemma, we can find a minimum weight branching in G^D. Next, assume that despite the addition of the edges in Aug to G_0, the resulting graph has bridges. All such bridges are the tree edges in Γ. Let $(u, p(u))$ be one such edge of Γ that is closest to the root (it does not have to be unique). Since vertices in the subtree rooted at u, are reached from r in the branching it must be the case that there is a directed edge $w \rightarrow v$, from a vertex w (ancestor of u) to v in the minimum weight branching. Such an edge would have been generated by an edge connecting v to a vertex not in the subtree rooted at u. This edge would belong to Aug and hence the edge $(u, p(u))$ is not a bridge. □

Lemma 3.3 *The weight of Aug is less than twice the optimal augmentation. (Weight$(Aug) \leq 2C^*$ where C^* is the optimal augmentation weight.)*

Proof: We prove the lemma by exhibiting a branching whose weight is at most twice the weight of the optimal augmentation. Consider the minimum weight set of edges Aug^* that would increase the connectivity from 1 to 2. Consider all the directed edges that are "generated" by edges that belong to Aug^*. These directed edges together with the tree edges yield a strongly connected graph with total weight on the edges at most $2C^*$ (each edge of weight w generated at most two directed edges, each of weight w). Hence the branching that we find has total weight at most $2C^*$. □

Theorem 3.4 *There is an approximation algorithm to find an augmentation to increase the edge connectivity of a connected graph to 2 with weight less than twice the optimal augmentation that runs in $O(m + n \log n)$ time.*

Proof: The correctness of the algorithm follows from Lemma 2 and Lemma 3. Since the bridge-connected components can be found in $O(m+n)$ time [AHU] and a minimum weight branching can be found in $O(m + n \log n)$ time [GGST86]. Since the least common ancestors for the m pairs can be found in $O(m + n)$ time by using the algorithm of Harel and Tarjan [HT84], the theorem follows. □

The approximation factor of 2 for the algorithm is tight. In [FJ81] an example is given to show that their algorithm can actually result in an approximation that is twice the optimal solution. The same example works for our algorithm as well.

4 Increasing Vertex Connectivity from 1 to 2

We can assume w.l.o.g. that G_0 is a connected graph just as in the case of edge connectivity. Our overall strategy is similar to the one used in the previous section. That is, we first obtain a tree structure Γ of the blocks of G_0, construct a weighted, directed graph G^D using Γ and G. Then find a minimum weight branching in G^D which will indicate the edges of $E - E_0$ that are to be added to increase the connectivity of G_0. We remark that Γ, in the case of vertex connectivity, is quite different from that of the previous section.

Algorithm *Construct Block Cut Tree*
Input: Connected graph G_0.
Output: Block cut tree Γ of G_0.

(1) Let $a_1, a_2, ...$ and $B_1, B_2, ...$ be the articulation points and blocks of $G_0 = (V, E_0)$, respectively. The vertex set $V(\Gamma)$ is a union of V_a and V_b where $V_a = \{a_1, a_2, ...\}$ and $V_b = \{b_i \mid B_i$ is a block of $G_0\}$. Associated with each vertex in $V(\Gamma)$, is a set. For $a_i \in V_a$, $X_i = \{a_i\}$. For $b_i \in V_b$, $Y_i = \{v_j \mid v_j \in V$ and v_j is not a cut vertex in $G_0\}$.

(2) The edge set $E(\Gamma)$ consists of edges (a_i, b_j) where a_i is an articulation point that belongs to block B_j.

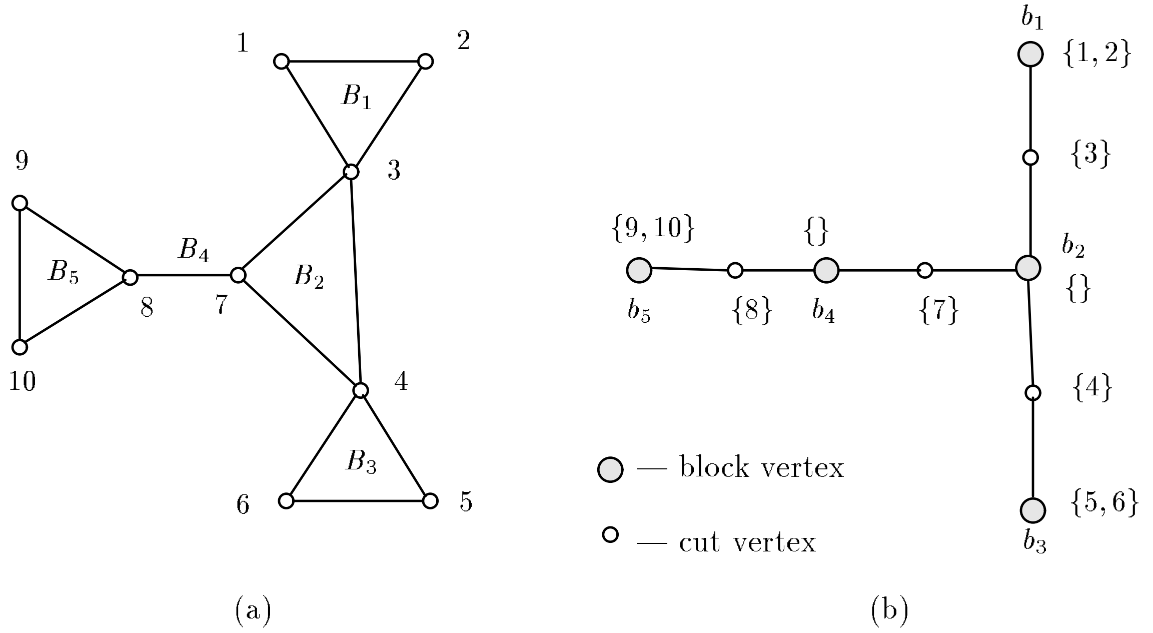


Figure 2: Construction of block cut vertex tree Γ.

Figure 2 illustrates the above construction via an example.

Observation 4.1 *In the block cut tree Γ, each edge is between a vertex in V_a and a vertex in V_b.*

Observation 4.2 *Consider the sets associated with the vertices of Γ. Each vertex of G_0 belongs to exactly one such set.*

In the rest of the section, for a vertex u of V, the vertex of Γ that *corresponds* to u is u if u is an articulation point, and b_i otherwise where B_i is the unique block containing u. In the following, by *superimposing* an edge $(x, y) \in G$ on Γ, we mean adding an edge between $a, b \in V(\Gamma)$ where the associated sets X and Y contain x and y respectively.
Algorithm *Find Vertex Aug*
Input: Graph G, a connected subgraph G_0, and a block cut tree Γ of G_0.
Output: A set of edges $Aug \subseteq E - E_0$.

(1) Superimpose all the edges of $E - E_0$ on Γ. Discard all the self-loops. Among the multiple edges retain the cheapest edge, discarding the rest.

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

(a) Pick an arbitrary leaf of Γ to be the root r, and direct all the edges of Γ towards r. Continue to denote the resulting tree by Γ.

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

(c) Consider the superimposed edges of $E - E_0$ on Γ. Let (u, v) be one such superimposed edge. If (u, v) is a back edge (i.e. it connects a vertex to one of its ancestors), we add one directed edge to E^D (shown below); otherwise, we add four directed edges to E^D. (We will refer to these directed edges as *images* of (u, v), and we say these directed edges are *generated* by (u, v).)

Suppose that the edge e with weight w, joins vertices u and v belonging to the tree Γ. There are two cases depending on the relative locations of u and v in the tree Γ. (See Figure 3.)

(i) If u is an ancestor of v (the other case is symmetric): then add an edge $u \rightarrow v$ in G^D with weight w.

(ii) If neither u nor v is an ancestor of the other: let $t = l.c.a(u, v)$ (least common ancestor in the rooted tree Γ). Add edges $t \rightarrow u$ and $t \rightarrow v$ in G^D, each with weight w. Also add edges $u \rightarrow v$ and $v \rightarrow u$, each with weight w.

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

(3) Find a minimum weight branching in G^D rooted at r. For each directed edge e that is picked as part of the branching, and does not belong to the directed tree Γ, add the corresponding edge in $E - E_0$ that generated e. The set of edges added is Aug.

Observation 4.3 *In the directed graph G^D there are no outgoing edges from a cut vertex to any of its descendants in Γ.*

Observation 4.4 *Consider the components formed on the deletion of a vertex $u \in V_a$ from Γ. The edges of G when superimposed on $\Gamma - u$ connect all these components.*

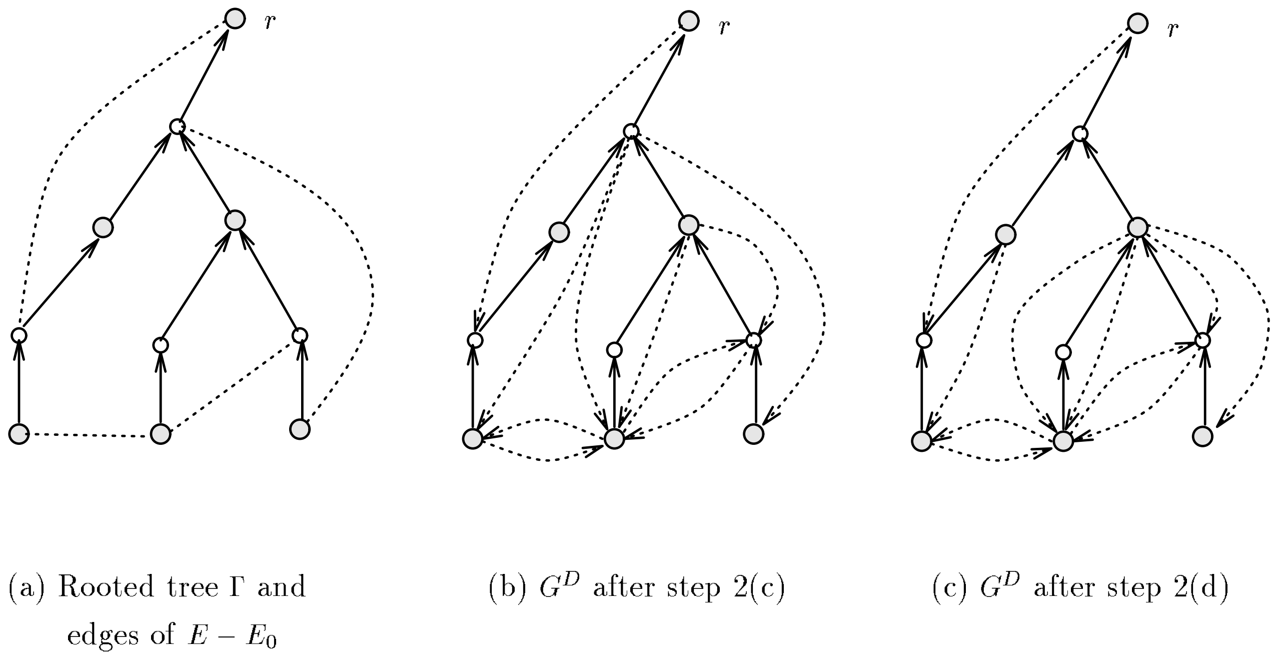
Lemma 4.5 *If G is 2-vertex connected, then the directed graph G^D is strongly connected.*

Proof: Clearly all the vertices of G^D can reach the root r using edges of the tree Γ. Let us assume that G^D is not strongly connected. Of all the vertices that cannot be reached from the root, let u be a vertex that is closest to the root in Γ. Clearly, the entire subtree rooted at u must consist of unreachable vertices and every proper ancestor of u is reachable from r. The proof is a little involved and we break it into cases.

Case 1: $u \in V_a$.

Since u is not a cut vertex in G, there must be at least one edge connecting a vertex in $C_1(u)$ to some vertex in $C_i(u)$ by Observation 4.4. Let this edge be (v, s) where $s \in C_1(u)$ and $v \in C_i(u)$. Now there are two subcases to consider:

(a) s is an ancestor of v.



r
r
r
(a) Rooted tree Γ and edges of $E - E_0$
(b) G^D after step 2(c)
(c) G^D after step 2(d)

Figure 3: Construction of G^D in the case of vertex connectivity.

(i) $s \in V_a$.
Corresponding to edge (s, v) we added an edge in G^D, from the child s_v of s (on the path from s to v) to v. Since s_v is a block vertex, it is distinct from u. Clearly s_v is an ancestor of u, and hence reachable from r. Thus v is reachable from r and so is u, yielding a contradiction.

(ii) $s \in V_b$.
We add an edge in G^D from s to v. Since s is reachable from r (because it is an ancestor of u), so is v and hence u, yielding a contradiction.

(b) s is not an ancestor of v. Let $t = l.c.a(s, v)$.

(i) $t \in V_a$.
Corresponding to edge (s, v) we added an edge in G^D, from the child t_v of t (on the path from t to v) to v. Since $t_v \in V_b$, it is distinct from u. It is reachable from r and hence v is reachable from r, and so is u, yielding a contradiction.

(ii) $t \in V_b$.
Clearly t is an ancestor of u, hence reachable from r. We added an edge in G^D from t to v, hence v is reachable and u as well, yielding a contradiction.

Case 2: $u \in V_b$.
Consider the cut vertex $p(u)$. Notice that $p(u) \neq r$ since $r \in V_b$. Let the roots of the subtrees $C_1(p(u)), C_2(p(u)), \ldots C_k(p(u))$ be $r_1(= r), r_2, \ldots r_k$, where k is the degree of $p(u)$. Assume that $C_2(p(u))$ refers to the component containing u (hence $r_2 = u$). Partition the components into two groups as follows. The first group contains all the components whose roots are reachable from r, and the second group contains the rest. (Notice that both the groups are non-empty.) Since G is biconnected there must exist an edge (s, v) where s belongs to a vertex in $C_i(p(u))$ and v belongs to a vertex in $C_j(p(u))$, such that $C_i(p(u))$

and $C_j(p(u))$ belong to the first and second groups respectively.

(a) s is an ancestor of v.

(i) $s \in V_a$.
We added an edge from the child s_v of s to v in G^D. Since s_v is a block vertex, it is distinct from $p(u)$ and reachable from r. Hence v is reachable from r, and so is r_j giving a contradiction.

(ii) $s \in V_b$.
We added an edge in G^D from s to v. Since s is an ancestor of $p(u)$ it is reachable from r. Hence v is reachable from r, and so is r_j, giving a contradiction.

(b) s is not an ancestor of v. Let $t = l.c.a(s, v)$.

(i) $t \neq p(u)$.
There is an edge in G^D from either t or t_v, to v. Since both t and t_v are reachable from r, so is v and hence r_j, giving a contradiction.

(ii) $t = p(u)$.
Note that r_i is reachable from r. Because of edge (s, v) we generate the following edges in G^D: $r_i \rightarrow s, r_j \rightarrow v, s \rightarrow v, v \rightarrow s$. Hence v is reachable from r, and so is r_j, yielding a contradiction.

□

Lemma 4.6 *If G is 2-vertex connected, then the vertex connectivity of the graph G_0 together with the edges in Aug is at least 2.*

Proof: Assume that despite the addition of the edges in Aug to G_0, the resulting graph has a cut vertex u. We will now show that u is destroyed as a cut vertex in the tree Γ, and hence in G_0. Consider the components $C_1(u), \ldots, C_{d(u)}(u)$ in Γ. Partition the components into two groups as follows. The first group contains all the components that get connected to $C_1(u)$ (by an edge or a path) when the edges of Aug are superimposed on Γ. The second group contains the rest. Notice that both the groups are non-empty. Since G^D is strongly connected all the vertices are reachable from the root in the minimum weight branching. Since there are no outgoing edges from u to its descendants by Observation 4.3, these must be an edge $s \rightarrow v$ in the branching that satisfies the following. This edge has the property that $s \in C_i(u)$ and $v \in C_j(u)$, where $C_i(u)$ and $C_j(u)$ belong to the first and second groups respectively. The edge that generated $s \rightarrow v$ in Aug would connect $C_i(u)$ to $C_j(u)$ in $G_0 + Aug$, yielding a contradiction. □

Lemma 4.7 *The weight of Aug is less than twice the optimal augmentation. (Weight(Aug) $\leq 2C^*$ where C^* is the optimal augmentation weight.)*

Proof: We prove the lemma by exhibiting a branching whose weight is at most twice the weight of the optimal augmentation. Consider the minimum weight set of edges Aug^* that would increase the connectivity from 1 to 2. Consider all the directed edges that are "generated" by edges that belong to Aug^*. These directed edges together with the tree edges yield a strongly connected graph with total weight on the edges at most $4C^*$ (each edge of weight w_i generated at most four directed edges, each of weight w_i). Now pick a minimum weight branching in this graph. Notice that for each cross edge (u, v) (when neither u nor v is an ancestor of the other) even though we generate four directed edges in G^D, no minimum weight branching will use more than two of these four edges. (Otherwise, it will not be a valid branching.) Hence the branching that we find has total weight at most $2C^*$. □

Theorem 4.8 *There is an approximation algorithm to find an augmentation to increase the vertex connectivity of a connected graph to 2 with weight less than twice the optimal augmentation that runs in $O(m + n \log n)$ time.*

Proof: The correctness of the algorithm follows from Lemma 5 and Lemma 6. Since the biconnected components can be found in $O(m+n)$ time [AHU] and a minimum weight branching can be found in $O(m + n \log n)$ time [GGST86]. Since the least common ancestors for the m pairs can be found in $O(m + n)$ time by using the algorithm of Harel and Tarjan [HT84], the theorem follows. □

5 Increasing Edge Connectivity to k

In this section, we show that it is possible to obtain an approximation factor of 2 for increasing the edge connectivity of a graph to any k. The algorithm takes as input an undirected graph $G_0(V, E_0)$ on n vertices and a set *Feasible* of m weighted edges on V, and finds a subset Aug of edges which when added to G_0 make it k-edge connected. The weight of Aug, is no more than twice the weight of the least weight subset of edges of *Feasible* that increases the connectivity. We also observe that the problem is NP-hard (for any k) by extending the proof that was given by [FJ81] for incrementing 1-connected graphs to 2-connected optimally.

Consider a directed graph G with weights on the edges, and a fixed root r. How does one find the *cheapest* directed subgraph H^D that has k-edge disjoint paths from a fixed root r to each vertex v ? Gabow [G91a] gives the fastest implementation of a weighted matroid intersection algorithm to solve this problem in $O(kn(m + n \log n) \log n)$ time. (See also [Ed79, FT89].)

To solve our problem (approximation algorithm), in the undirected graph G_0 replace each undirected edge (u, v) by two directed edges $u \rightarrow v$ and $v \rightarrow u$ with each edge having weight 0. For each edge in the set *Feasible* (u, v) we replace it by two directed edges $u \rightarrow v$ and $v \rightarrow u$ with weight $w(u, v)$ (the weight of the undirected edge). Call this graph G^D. Now run Gabow's algorithm on the graph G^D, asking for k-edge disjoint paths from each vertex to the root. If the directed edge $u \rightarrow v$ is picked in H^D and $w(u, v) > 0$ (we can assume all edges of set *Feasible* have weight > 0 else we can always include it in Aug) we add (u, v) to *Feasible*. (This is a generalization of the scheme in [KV92], where it was shown only for the case when E_0 is empty.)

Lemma 5.1 *The graph $E_0 + Aug$ is a k-edge connected graph.*

Proof: Suppose there is a cutset C of $(k - 1)$ edges in $G_k = G_0 + Aug$. Assume that it separates G_k into components C_1 and C_2. Let r be in C_1. Consider a vertex v in C_2. It cannot be that there were k-edge disjoint paths from r to v in G^D (because of the cutset C). □

Theorem 5.2 *The total weight of* Aug *is at most twice the weight of the optimal augmentation.*

Proof: Consider the optimal augmentation Aug^*. Consider the following subgraph: add the anti-parallel edges corresponding to edges in E_0 and Aug^*. In this graph clearly there are k-edge disjoint (directed) paths from r to each vertex v (by directing the undirected paths from r to v appropriately). Thus the optimal solution to the problem (Aug) must have total cost less than 2 Aug^*. □

6 Increasing Edge Connectivity from k to $k+1$ (for odd k)

The method described in Section 3 can be applied in a more general setting: to find an approximation to the problem of increasing the edge connectivity of a k-edge connected graph (say $G_k = (V, E_k)$) to $k+1$ when k is odd. The key idea is to use the representation of cuts that is due to Karzanov and Timofeev [KT86]. We note that this structure was used by Naor, Gusfield and Martel [NGM90] to solve the augmentation problem exactly for the unweighted case when *Aug* allows an edge to be added between any pair. The structure itself can be constructed in $O(kn^2)$ time. We briefly describe this structure in the following. In this structure each vertex of G_k maps to exactly one vertex in the representation, i.e., each vertex of the representation corresponds to a subset (possibly empty) of vertices of G_k. Moreover, the cuts of a graph can be represented by a tree when the connectivity of G_k is odd; otherwise, the structure is a "tree-like" structure. Denote this structure by τ. The cut information about G_k is encoded as follows. When k is odd, the number of cuts in a k-edge connected graph is $O(n)$ [DKL76]. Assume that removing a cut (i.e., a set of k edges whose removal disconnects the graph) from G_k results in two components H_1 and H_2 where $V(G_k) = V(H_1) \cup V(H_2)$. In τ there exists an edge whose removal results in two subtrees with the following property: the union of the subsets of the vertices represented by the vertices of one of the subtrees is $V(H_1)$ and of the other is $V(H_2)$.

To increase the connectivity of G_k, construct τ and a set of edges F on the vertices of τ that correspond to the set of edges *Feasible*. That is, for every $(u,v) \in$ *Feasible*, include an edge in F between the unique pair of vertices of τ that corresponds to u, v. Now, run the algorithm *Find Edge Aug* with $\tau + F$ for G and τ for G_0. At the end of the algorithm, pick the subset of the edges (denote it as Aug') of *Feasible* that correspond to the edges of the output *Aug*. It follows from Theorem 3.4 and the fact the τ can be build in $O(kn^2)$ time that the time complexity of the entire process is $O(kn^2)$. We establish the correctness of our method.

Lemma 6.1 *The graph $E_k + Aug'$ is a $(k+1)$-edge connected graph.*

Proof: Assume that there is a cutset C of k edges in $E_k + Aug'$. Notice that all the edges of C must come from E_k as we assumed that G_k is k-edge connected. Assume that C separates the augmented graph into components H_1 and H_2. The cut tree representation guarantees that there is a tree edge in τ that corresponds to C. Denote the two subtrees resulting from removing this tree edge by τ_1 and τ_2. Since this tree edge is not a bridge in $\tau + Aug$, by Lemma 3.2 there is an edge of *Aug* that connects a vertex of τ_1 to a vertex of τ_2. There is an edge in Aug' that corresponds to this edge of *Aug*. But this edge must connect a vertex of H_1 to a vertex of H_2 contradicting the assumption that H_1 is separated from H_2 when C is deleted from $G_k + Aug'$.
$\square$

The next theorem follows from Lemma 3.3 and the observation that when the edges corresponding the subset of edges from the optimal augmentation are added to τ, the resulting graph is 2-edge connected.

Theorem 6.2 *The total weight of* Aug *is at most twice the weight of the optimal augmentation.*

Acknowledgments: We thank Professors Joseph JáJá and Uzi Vishkin for encouragement and useful discussions. We would also like to thank the referees for useful comments on an earlier draft.

References

[AHU] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[DKL76] E.A. Dinits, A.V. Karzanov and M.L. Lomosonov, "On the structure of a family of minimal weighted cuts in a graph," *Studies in Discrete Optimization* [In Russian], A.A. Fridman (ear decomposition), Nauka, Moscow, pp. 290-306, 1976.

[Ed79] J. Edmonds, "Matroid intersection," *Annals of Discrete Mathematics*, No. 4, pp. 185–204, (1979).

[Ev79] S. Even, *Graph Algorithms*, Computer Science Press, Potomac, Md., 1979.

[ET76] K. P. Eswaran and R. E. Tarjan, "Augmentation problems," *SIAM Journal on Computing*, Vol. 5, No. 4, pp. 653–665, (1976).

[Fr90] A. Frank, "Augmenting graphs to meet edge-connectivity requirements," *31st Annual Symposium on Foundations of Computer Science*, pp. 708–718, (1990).

[FJ81] G. N. Frederickson and J. JáJá, "Approximation algorithms for several graph augmentation problems," *SIAM Journal on Computing*, Vol. 10, No. 2, pp. 270–283, (1981).

[FJ82] G. N. Frederickson and J. JáJá, On the relationship between the biconnectivity augmentation and traveling salesman problems," *Theoretical Computer Science*, Vol. 19, No. 2, pp. 189–201, (1982).

[FT89] A. Frank and E. Tardos, "An application of submodular flows," *Linear Algebra and its Applications*, 114/115, pp. 320–348, (1989).

[G91a] H. N. Gabow, "A matroid approach to finding edge connectivity and packing arborescences," *23rd Annual Symposium on Theory of Computing*, pp. 112–122, (1991).

[G91b] H. N. Gabow, "Applications of a poset representation to edge connectivity and graph rigidity," *32nd Annual Symposium on Foundations of Computer Science*, pp. 812–822, (1991).

[GGST86] H. N. Gabow, Z. Galil, T. Spencer and R. E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs," *Combinatorica*, 6 (2), pp. 109–122, (1986).

[HR91a] T. S. Hsu and V. Ramachandran, "A linear time algorithm for triconnectivity augmentation," *32nd Annual Symposium on Foundations of Computer Science*, pp. 548–559, (1991).

[HR91b] T. S. Hsu and V. Ramachandran, "On finding a smallest augmentation to biconnect a graph," 2^{nd} *Annual International Symposium on Algorithms*, Springer Verlag LNCS 557, pp. 326–335, (1991).

[HT84] D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM Journal on Computing*, Vol 13, No. 2, pp. 338–355, (1984).

[KB91] G. Kant and H. Bodlaender, "Planar Graph Augmentation Problems," *1991 Workshop on Algorithms and Data Structures*, pp. 286–298, (1991).

[KT86] A.V. Karzanov and E.A. Timofeev, "Efficient algorithm for finding all minimal edge cuts of a nonoriented graph," *Cybernetics*, pp. 156-162, Translated from *Kibernetika*, No. 2, pp. 8–12, (1986).

[KV92] S. Khuller and U. Vishkin, "Biconnectivity approximations and graph carvings," Technical Report UMIACS-TR-92-5, CS-TR-2825, Univ. of Maryland, January (1992), Also to appear in *24th Annual Symposium on Theory of Computing*, (1992).

[NGM90] D. Naor, D. Gusfield and C. Martel, "A fast algorithm for optimally increasing the edge-connectivity," *31st Annual Symposium on Foundations of Computer Science*, pp. 698–707, (1990).

[RG77] A. Rosenthal and A. Goldner, "Smallest augmentations to biconnect a graph," *SIAM Journal on Computing*, Vol. 6, No. 1, pp. 55–66, (1977).

[Wa88] T. Watanabe, "An efficient augmentation to k-edge connect a graph," Tech. Report C-23, Department of Applied Math., Hiroshima University, April 1988.

[WN87] T. Watanabe and A. Nakamura, "Edge-connectivity augmentation problems," *Journal of Comp. and Sys. Sciences*, 35 (1), pp. 96–144, (1987).