When cycles collapse: A general approximation technique for constrained two-connectivity problems

R. Ravi and Philip Klein

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-92-30
June 1992

When cycles collapse: A general approximation technique for constrained two-connectivity* problems

R. Ravi[1]
Philip Klein[2]
Brown University

Abstract

We present a general approximation technique for a class of network design problems where we seek a network of minimum cost that satisfies certain communication requirements and is resilient to worst-case single-link failures. Our algorithm runs in $O(n^2 \log n)$ time on a graph with n nodes and outputs a solution of cost at most thrice the optimum. We extend our technique to obtain approximation algorithms for augmenting a given network so as to satisfy certain communication requirements and achieve resilience to single-link failures.

Our technique allows one to find nearly minimum-cost two-connected networks for a variety of connectivity requirements. For example, our result generalizes earlier results on finding a minimum-cost two-connected subgraph of a given edge-weighted graph in [3, 9] and an earlier result on finding a minimum-cost subgraph two-connecting a specified subset of the nodes in [14]. Using our technique, we can also approximately solve for the first time a two-connected version of the generalized Steiner network problem and a two-connected version of the non-fixed point-to-point connection problem.

1 Introduction: The framework

On designing a network

The following scenario was proposed in [1]. You are in the job of providing communication links to customers. You have a set of clients with communication requirements; each client has specified a pair of cities (called a *site-pair*) between which the client must have communication capabilities. In front of you is the AT&T[3] price list, which gives prices for constructing communication links between various cities. Each link built has essentially infinite communication bandwidth. Your job is to select a minimum-cost collection of communication links that can accommodate all your clients' communication requirements. The network you must construct need not be connected; all that is needed is that every client's pair of cities be connected through your network. Unfortunately, your job's NP-complete. So you come up with an approximately good solution [1] and build a network using this strategy.

Soon you notice that your clients are complaining. The links used in your network seem to be breaking down pretty often. And you notice that it is very often the case that it is just a single edge that has caused each tree in your solution to come to a standstill. So your clients want you to augment this network so that each pair of cities have at least two connections. Moreover, they do not want you to duplicate any link in the already existing network since the link failures seem to be related to the location of the cables. By duplicating links, you are only making *both* copies prone to failure. Well, your problem is NP-complete again [3]. If you were lucky to have a large enough clientèle so that your network spanned all the cities, then you could use the approximate solution in [3]. Or if you were plain lucky and

*Connectivity refers to edge-connectivity throughout this paper unless stated otherwise explicitly.

[1]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.

[2]Research supported by NSF grant CCR-9012357 and NSF PYI award CCR-9157620, together with PYI matching funds from Thinking Machines Corporation and Xerox Corporation. Additional support provided by DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

[3]Disclaimer: Even now, the authors are in no way connected to AT&T.

your network had only one tree, you could turn to the extension in [14] to augment this tree. However, considering your past history with fortune, no approximation algorithms are known in the general case.

In this paper, we give the first approximation algorithm for augmenting this generalized Steiner network to a network in which each site-pair is two-connected. Combining this with the solution offered to your earlier problem in [1], this provides an approximation for a general two-connected Steiner network design problem. In fact, you weren't so unlucky after all. For, while you set out to solve the problem of finding a minimum cost network two-connecting all the site-pairs, your technique turns out to solve a whole class of minimum cost two-connected subgraph problems...

An example of a solution formed this way for the generalized two-connected Steiner network problem is depicted in Figure 1.

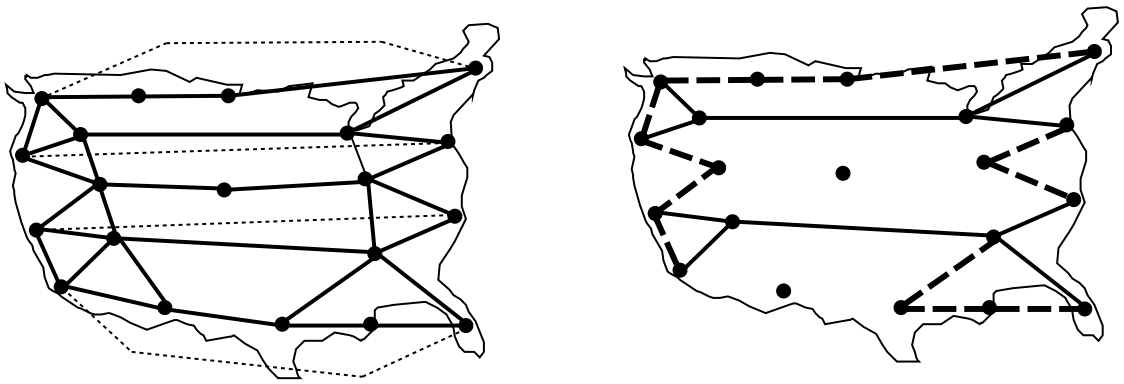

Figure 1: An instance of the unweighted generalized two-connected Steiner network problem and an optimum solution. Solid edges correspond to unit-cost links; dotted edges connect site pairs. In the right figure, the solid edges correspond to first phase one-connecting solution. The thick dashes form an augmentation to two-connect the solid trees.

Framework

We can model the generalized two-connected Steiner network problem as follows: Define a cut in the graph to be *active* if it separates some site-pair. A network is said to *two-connect* a site-pair if there are two edge-disjoint paths between them in the network. Thus, any network that two-connects the site-pairs must cross this active cut twice. Conversely, if a network crosses every such active cut twice, the network indeed two-connects all the site-pairs. More formally, we can define a function f defining the activity of all cuts in the graph. If a cut C separates some site-pair, then we set $f(C) = 1$ and call C an *active* cut. Otherwise, we set $f(C) = 0$ and call C an *inactive* cut. Any set of edges that crosses each of the active cuts twice is a feasible solution to the two-connected Steiner network problem. We can generalize this two-connected network design problem by extending our notion of active cuts. For this, we turn to the work of Goemans and Williamson in [6] where they define the notion of *proper* functions. Proper functions are a class of functions used for defining the activity of cuts in a graph.

In [1], we provided the first approximation algorithm for the generalized network Steiner problem. The algorithm presented there involved construction of feasible primal and dual solutions with an approximate relation between their values. Subsequently, Goemans and Williamson [6] generalized this result to a technique for approximating a large class of constrained forest problems. They introduced the notion of proper functions whose elegant definition captures the connectivity requirements for a variety of constrained forest problems. In this paper, we use their notion of *proper functions* in formulating a general two-connected network design problem.

Given a graph $G = (V, E)$, a function $f : 2^V \rightarrow \{0, 1\}$ and a non-negative cost function c on the edges, we consider the following integer program:

Min $\sum_{e \in E} c_e x_e$
subject to constraints:

$$\begin{array}{rcll} x(\Gamma(S)) & \geq & 2f(S) & \emptyset \neq S \subset V \qquad \text{(IP)} \\ x_e & \in & \{0,1\} & e \in E \end{array}$$

Here $\Gamma(S)$ denotes the set of edges having exactly one endpoint in the set S and $x(F) = \sum_{e \in F} x_e$. Any feasible solution to the program above is a collection of edges that intersect each active cut under the function f at least twice. Thus, any minimal feasible solution to (IP) is a collection of two-connected subgraphs. In this paper, we address solutions to the above program for the class of proper functions f. An important requirement of the solution subgraph that this paper addresses is that edges are not allowed to be replicated. This will be the assumption in all the problems we consider. Thus, the program is feasible only if the input graph has a feasible solution as a subgraph. We shall assume that the input graph is itself feasible for (IP). It is not hard to verify this assumption. We define A, the set of *terminals* in V as the set $A = \{v \in V \ : \ f(\{v\}) = 1\}$. The following is our main result.

Theorem 1.1 *There is an $O(n^2 \log n)$ algorithm to produce an feasible solution F' to (IP) such that $\sum_{e \in F'} c_e \leq (3 - \frac{3}{|A|})OPT$ where OPT is the optimum value of (IP).*

Although we use weak linear programming duality in our analysis, our algorithm does not linear programming.

Formulating problems in this framework - Examples

The generalized two-connected Steiner network problem without edge replication is representable in the above framework. We set $f(S) = 1$ if there is a site pair that contains a site in S and one outside S, and we set $f(S)$ to 0 otherwise. The above framework also captures the following *point-to-point two-connection problem* we introduce. In this, we are given a set $C = \{c_1, \ldots, c_p\}$ of sources and a set $D = \{d_1, \ldots, d_p\}$ of destinations in the graph and we need to find a minimum-cost set F of edges such that each source-destination pair is two-connected in F. The fixed destination case, in which c_i is required to be two-connected to d_i, reduces to a case of the generalized two-connected Steiner network problem. However, we can also approximate the non-fixed destination case, where the only requirement is that each component of F contains the same number of sources and destinations. For this, we set $f(S)$ to 1 whenever $|S \cap C| \neq |S \cap D|$ and to 0 otherwise.

Strategy

The algorithm we present in this paper works in two phases. In the first phase, the algorithm in [6] is applied to construct a partial solution F_1 that partially satisfies the requirements in (IP), namely, the solution satisfies the one-connectivity constraints: $x(\Gamma(S)) \geq f(S) \quad \emptyset \neq S \subset V$. In the second phase, we augment the partial solution F_1 such that the augmented subgraph two-connects each of the trees in the forest F_1. We show that the cost of the solution output at the first phase is at most the optimum value of (IP) and the cost of the augmentation output at the second phase is at most twice the optimum value of (IP) to obtain a performance guarantee of three. The second phase is a careful reapplication of the methods of Goemans and Williamson. We define an auxiliary function f' specifying a different notion of active cuts, based on the partial solution F_1. If f' were proper, then we could reapply the method in [6] directly. Unfortunately, f' is not proper and so we need to adapt this method.

Approximating augmentation

The techniques used in proving Theorem 1.1 can be adapted to provide approximation algorithms for minimum-cost augmentation. Suppose that we are given a graph $G = (V, E)$ along with an initial subgraph G_0 with edge set E_0. The *minimum-cost augmentation problem* is to choose a minimum-cost set of edges in G whose addition to G_0 yields a graph that obeys the constraints of (IP). This problem generalizes the minimum-cost bridge-connectivity augmentation problem considered in [3, 9] and also the Steiner version considered in [14]. We have the following theorem about approximating the minimum-cost augmentation.

Theorem 1.2 *The augmentation problem described above can be approximated in $O(n^2 \log n)$ time within a factor of $2(1 - \frac{1}{|A|})$ if the incidence vector, x_0 of the initial subgraph G_0 obeys the one-connectivity constraints*

$$x_0(\Gamma(S)) \geq f(S) \quad \forall S \ : \ \emptyset \neq S \subset V$$

Otherwise the performance guarantee is $3(1 - \frac{1}{|A|})$.

Road-map

In the next section, we look at related work. Thereafter, we present background results and the basic technique used in proving our main result. In Section 4, we present the second-phase augmentation algorithm. In the following section, we prove the performance bound of this phase. Then we prove the results on augmentation. We close with some implementation issues and open problems.

2 Related work

Fredrickson and Ja'Ja', in their pioneering work [3] on approximation algorithms for weighted graph augmentation problems, provided the first approximate solution for the problem of augmenting a given graph to be two-connected. The performance guarantee of their algorithm was a multiplicative factor of two. Their technique also gave an approximate solution to the minimum-cost two-connected subgraph problem. The performance guarantee is a multiplicative factor of three. Their solution for the latter problem starts with a minimum spanning tree and augments this to be two-connected. Khuller and Thirumella [9] have extended these results and provided more efficient algorithms for the same. Recently, Ravi [14] observed that this method generalizes to find minimum-cost two-connected Steiner subgraphs of a given graph. All the above methods are also able to solve the node two-connected versions approximately. Khuller and Vishkin have presented algorithms with better performance guarantees for the minimum two-connected subgraph problems in [10]. The bounds are $\frac{3}{2}$ for the two edge-connected case and $\frac{5}{2}$ for the two node-connected case.

When the cost function c obeys the triangle inequality, then Fredrickson and Ja'Ja' [4] present an adaptation of Christofides' heuristic to solve the minimum-cost biconnectivity augmentation problem with a performance factor of $\frac{3}{2}$. There has been considerable previous work on network survivability [5, 8, 11, 12, 15] that addresses constructing minimum-cost k-connected subgraphs under such cost functions. However, our interest is in the general case when the cost function does not satisfy the triangle inequality.

There has also been considerable work on characterizations of integer polyhedra arising from connectivity constraints [7], and construction of minimum-cost two-connected survivable networks [5, 13].

As mentioned in the previous section, Agrawal, Klein and Ravi [1] introduced the technique of a primal-dual method for approximating the minimum-cost generalized Steiner network. Though this result addressed arbitrary connectivity requirements between pairs of nodes, it allowed for edge replication. This was consequently generalized by Goemans and Williamson in [6].

Our paper combines the two-phase idea of Fredrickson and Ja'Ja' with a primal-dual method tailored to provide an approximate solution in both phases.

3 Background

Following [6], we define the notion of a proper function $f \ : 2^V \rightarrow \{0, 1\}$. A function f is *proper* if the following properties hold.

- [Null] $f(\emptyset) = 0$;
- [Symmetry] $f(S) = f(V - S)$ for all $S \subseteq V$; and
- [Disjointness] If A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$.

The functions f used to model the problems in the first section can be easily verified to be proper.

The first phase:

As mentioned in the introduction, our algorithm works in two phases. In the first phase, we construct an approximate solution to the following modified integer program:

$$\begin{array}{l}\text{Min } \sum_{e\in E} c_e x_e \\ \text{subject to constraints:} \\ \begin{array}{rcll} x(\Gamma(S)) & \geq & f(S) & \emptyset \neq S \subset V \qquad \text{(IP1)} \\ x_e & \in & \{0,1\} & e \in E \end{array}\end{array}$$

We do this using the method in [6]. Let the solution generated be denoted by F_1. We shall use F_1 to denote the *set of edges* in this solution returned in the first phase. It follows from [6] that F_1 induces a forest in G. The algorithm in [6] implicitly constructs a dual solution to the above linear program to prove the performance guarantee. This dual to the linear relaxation of the above integer program (using $x_e \geq 0$) is the following.

$$\begin{array}{l}\text{Max } \sum_{S\subset V} f(S) \cdot y_S \\ \text{subject to constraints:} \\ \begin{array}{rcll} \sum_{S: e\in\Gamma(S)} y_S & \leq & c_e & e \in E \\ y_S & \geq & 0 & \emptyset \neq S \subset V \end{array}\end{array}$$

Recall that A is the set of terminals in V. We will be using the following results from [6].

Theorem 3.1 ([6]) *The solution F_1 feasible for (IP1) and the dual y constructed are related by*

$$\sum_{e\in F_1} c_e \leq (2 - \frac{2}{|A|}) \sum_{S\subset V} y_S$$

Lemma 3.2 ([6]) *Let $e \in N$ where N is a connected component of a minimal solution F_1. Let N_1 and N_2 be the two connected components of $N - \{e\}$. Then $f(N_1) = f(N_2) = 1$.*

Observation 3.3 ([6]) *If $f(S) = 0$ and $f(B) = 0$ for some $B \subseteq S$, then $f(S - B) = 0$.*

The LP relaxation of our integer program (IP) is obtained by replacing each integrality constraint $x_e \in \{0,1\}$ with the linear constraint $0 \leq x_e \leq 1$. The dual to this LP relaxation is as follows.

$$\begin{array}{l}\text{Max } \sum_{S\subset V} 2f(S) \cdot y_S - \sum_{e\in E} r_e \\ \text{subject to constraints:} \\ \begin{array}{rcll} \sum_{S: e\in\Gamma(S)} y_S & \leq & c_e + r_e & e \in E \\ y_S & \geq & 0 & \emptyset \neq S \subset V \qquad \text{(LP)} \\ r_e & \geq & 0 & e \in E \end{array}\end{array}$$

By weak linear programming duality, if OPT denotes the value of the optimal solution to (IP) and Z^*_{LP} denotes that of the optimal dual solution to (LP), then $OPT \geq Z^*_{LP}$. Note that the dual solution y provided by the algorithm of [6] is feasible for (LP) by setting $r_e = 0$ for all edges e. The value of the above program (LP) with this solution is $2\sum_{S\subset V} y_S$. Thus we have $OPT \geq 2\sum_{S\subset V} y_S$. Combining this with Theorem 3.1, we get the following bound on the first phase solution.

Lemma 3.4 *Let F_1 be the set of edges chosen in the first phase and let OPT denote the value of an optimum solution to (IP). Then*

$$\sum_{e\in F_1} c_e \leq (1 - \frac{1}{|A|})OPT$$

The second phase:

In the next section, we describe an algorithm for augmenting the forest F_1 with a set of edges F_2 disjoint from F_1. The algorithm also simultaneously constructs an implicit dual solution $\tilde{y}_S$ for $S \subset V$. We shall prove the following theorem in the next two sections.

Theorem 3.5 *The second phase algorithm chooses a set F_2 of edges disjoint from F_1, and an implicit dual solution $\tilde{y}$.*

(1) The set of edges in $F_1 \cup F_2$ two-connects each of the trees in (V, F_1).

(2) The solution $\tilde{y}$ will have the property that $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$. Also, the solution $\tilde{y}$ will obey the normal packing constraint on all edges not in F_1. Namely, for any edge $e \in E - F_1$, we have $\sum_{S:\Gamma(S) \ni e} \tilde{y}_S \leq c_e$.

We also prove the following relation, which is the basis for our performance analysis of the algorithm.

Theorem 3.6 *Let F_2 be the set of edges chosen by the second phase algorithm and let $\tilde{y}$ be the dual solution it implicitly constructs. Then*

$$\sum_{e \in F_2} c_e \leq (2 - \frac{2}{|A|}) \sum_{S \subset V} \tilde{y}_S$$

The strategy to prove a bound of two for the second phase is to show that $\tilde{y}$ is feasible to the dual (LP) above. For this, we need the following lemma.

Lemma 3.7 *If $|\Gamma(S) \cap F_1| = 1$ then $f(S) = 1$.*

Proof: Suppose $\Gamma(S) \cap F_1 = \{e\}$ and suppose $e \in N$, where $N, N_1, \ldots, N_m$ are components of (V, F_1). Let P and Q be the two components of $N - \{e\}$. Assume without loss of generality that $P \subseteq S$ and $Q \cap S = \emptyset$. Since $\Gamma(S)$ only intersects the edge e from F_1, for each N_i, either $S \cap N_i = \emptyset$ or $S \cap N_i = N_i$. Thus, we can write S as the disjoint union of P and some of the components in F_1. Now $f(N_i) = 0$ for all the N_i's. Also, from Lemma 3.2, we have $f(P) = 1$. Disjointness and Observation 3.3 together imply that $f(S) = 1$. □

Now, we show that $\tilde{y}$ is a feasible solution to (LP). Namely, for each edge $e \in F_1$, set $r_e = \sum_{S:\tilde{y}_S > 0,\ \Gamma(S) \cap F_1 = \{e\}} \tilde{y}_S$. Set $r_e = 0$ for all the other edges. Since we have $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$, we get

$$\sum_{e \in E} r_e = \sum_{e \in F_1} r_e = \sum_{e \in F_1} \sum_{S:\Gamma(S) \cap F_1 = \{e\}} \tilde{y}_S = \sum_{S:|\Gamma(S) \cap F_1| = 1} \tilde{y}_S = \sum_{S \subset V} \tilde{y}_S \tag{1}$$

Now, we show that the solution $\tilde{y}_S$ obeys all the packing constraints in (LP). For any edge $e \in F_1$, we have

$$\sum_{S:\Gamma(S) \ni e} \tilde{y}_S = r_e \leq r_e + c_e$$

since $c_e \geq 0$. Also, for any edge e not in F_1, we have

$$\sum_{S:\Gamma(S) \ni e} \tilde{y}_S \leq c_e \leq c_e + r_e$$

using the second part of Theorem 3.5. Finally, the value of the solution is

$$\sum_S 2f(S)\tilde{y}_S - \sum_e r_e \geq 2 \sum_{S:|\Gamma(S) \cap F_1| = 1} \tilde{y}_S - \sum_{e \in F_1} r_e = \sum_S \tilde{y}_S$$

The first inequality follows from Lemma 3.7 and the second equality from Equation (1). By weak linear programming duality, we again have $OPT \geq \sum_S \tilde{y}_S$. Combining this and Theorem 3.6, we get the following lemma.

Lemma 3.8 *Let F_2 be the set of edges chosen by the second phase algorithm and let OPT denote the value of an optimum solution to (IP). Then*

$$\sum_{e \in F_2} c_e \leq (2 - \frac{2}{|A|})OPT$$

We pause and prove that the solution $F_1 \cup F_2$ we offer is feasible.

Lemma 3.9 *$F_1 \cup F_2$ is feasible for (IP).*

Proof: For any $S \subseteq V$, we have $|\Gamma(S) \cap F_1| \geq f(S)$ since F_1 is feasible for (IP1). By Theorem 3.5, the set of edges in $F_1 \cup F_2$ two-connects all the trees in F_1. Thus if $f(S) = 1$ and $|\Gamma(S) \cap F_1| = 1$ then we must have $|\Gamma(S) \cap F_2| \geq 1$ for such S. Therefore if x is the incidence vector of $F_1 \cup F_2$ then for any S with $f(S) = 1$, we have

$$x(\Gamma(S)) = |\Gamma(S) \cap F_1| + |\Gamma(S) \cap F_2| \geq 2$$

Hence, x is feasible for (IP). □

We have described construction of a feasible solution $F_1 \cup F_2$ to (IP) and we have bounds on its value from Lemmas 3.4 and 3.8. These give us the Theorem 1.1. It remains to describe the algorithm to construct F_2 and the implicit dual $\tilde{y}_S$. We do this in the next section.

4 The algorithm

In this section, we describe the algorithm for augmenting a given forest F_1 such that each tree in F_1 becomes two-connected. Note that F_1 is a minimal solution to the integer program (IP1). We also implicitly construct a dual solution $\tilde{y}$ such that it has the properties we required in Theorem 3.5. Since we require that $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$, we define this condition as a new function f', namely,

$$f'(S) = \begin{cases} 1 & \text{if } |\Gamma(S) \cap F_1| = 1 \\ 0 & \text{otherwise} \end{cases}$$

It is important to note that the function f' defined this way is *not* proper. It can be easily shown to violate the disjointness requirement. Our basic algorithm for the second phase is similar to that of Goemans and Williamson but is modified to take care of this difference. As input to the algorithm, we are given an undirected graph G with edge costs $c_e \geq 0$ and the forest F_1. We output a set F_2 of edges such that $F_2 \cap F_1 = \emptyset$ and each tree in F_1 is two-connected in $F_1 \cup F_2$.

Overview

The algorithm maintains a forest, F, of edges as candidates for F_2. The forest F initially has no edges. It also maintains a set of *clusters* which partitions the vertex set at any time in the running of the algorithm. To begin with, each node in V is in its own cluster. The set of clusters at any time in the running of the algorithm can be partitioned into *active* and *dead* (or inactive) clusters, namely, those with f' value 1 and 0 respectively. The algorithm progresses in iterations, choosing edges between clusters and merging clusters. Each merge involves an active cluster. When no active cluster remains, the algorithm terminates. Whenever a chosen edge closes a cycle in $F_1 \cup F$, this cycle is collapsed: all the clusters containing nodes in the cycle are unioned into a single cluster. Note that the graph induced by $F_1 \cup F$ at the beginning of each iteration with the clusters contracted to single nodes is acyclic, because cycles are collapsed at each iteration. The algorithm also implicitly maintains a dual solution $\tilde{y}$. Only the active clusters are used to update the dual solution. As long as there remains an active cluster, there remain at least two clusters since $f'(V) = 0$. In the end, we do a clean-up to retain only essential edges and remove redundant edges.

The algorithm

We use the notation of [6]. We shall use $\mathcal{C}$ to denote the set of clusters that the algorithm maintains. For a vertex v, we let C_v denote the cluster containing v. We shall use a counter t which records the iteration number during the course of the algorithm. Henceforth, we shall use the term "at time t" to refer to iteration number t of the algorithm.

1 Initialize $F := \emptyset$, $\mathcal{C} := \{\{v\} \ : \ v \in V\}$, $t := 0$.

2 **Comment:** Implicitly set $\tilde{y}_S := 0$ for all $S \subseteq V$.

3 For each $v \in V$, set $d(v) := 0$.

4 While there is a $C \in \mathcal{C}$ such that $f'(C) = 1$

5 Find edge $e = (i, j)$ with $i \in C_1 \in \mathcal{C}$, $j \in C_2 \in \mathcal{C}$, $C_1 \neq C_2$ that minimizes the reduced cost $\delta = \frac{c_e - d(i) - d(j)}{f'(C_1) + f'(C_2)}$.

6 For all $v \in C \in \mathcal{C}$, set $d(v) := d(v) + \delta \cdot f'(C)$. **Comment:** Implicitly set $\tilde{y}_C := \tilde{y}_C + \delta \cdot f'(C)$.

7 Set $t := t + 1$ and $F := F \cup \{e\}$. **Comment:** Edge e is chosen at time t.

8 If e closes a cycle in $F_1 \cup F$, then union all the clusters containing nodes in this cycle to form a new cluster and insert this in $\mathcal{C}$. Delete each of the old clusters used in the unioning from $\mathcal{C}$. **Comment:** We say that a *collapse* has occurred at time t. All the edges in this cycle with their endpoints in different clusters are said to participate in this collapse.

9 If e does not close a cycle, union the two clusters C_1 and C_2 to form a new cluster and insert this in $\mathcal{C}$. Delete the two clusters C_1 and C_2 from $\mathcal{C}$.

10 (Clean-up step) Select a subset F_2 of F to output. **Comment:** This step is described later in this section.

Clean-Up

Before we describe the clean-up step, we need some notation. At any time t, let $\mathcal{C}(t)$ denote the set of clusters in $\mathcal{C}$ just *after* iteration number t of the algorithm. Let e_t denote the unique edge added in the t^{th} iteration. We define $F(t)$ to be set of edges in the set F added until time t including e_t. Let t_f be the total number of iterations before the algorithm terminates.

We shall refer to the graph (V, F_1) or any contracted version of it as the *skeleton*. Hence the edges in F_1 are termed *skeletal*. Also, a path made up of edges entirely in F_1 is called a *skeletal path*. Similarly, edges in F are termed *nonskeletal*.

Clean-up works by processing the edges in F in the *reverse* order in which they were chosen by the algorithm and discarding those that are not essential for two-connectivity. The procedure below takes as input the set of edges F chosen by the algorithm and outputs a subset $F_2 \subseteq F$ of edges to be retained in the final solution.

1 Initialize $F_2 := \emptyset$.

2 For $t := t_f$ down to 1 do

3 If the graph $(V, F_1 \cup F_2 \cup F(t) - \{e_t\})$ contains at least one skeletal bridge edge b_t, then set $F_2 := F_2 \cup \{e_t\}$.

Feasibility

We have the following observations from the algorithm.

Proposition 4.1 *At any iteration t of the algorithm, the set of nodes in each cluster in $\mathcal{C}(t)$ is one-connected using edges in $F_1 \cup F(t)$.*

The above proposition can be proved by induction on the iteration count t.
Definition: Define a skeletal edge e to be *enclosed* at time t if both its endpoints belong to the same cluster in $\mathcal{C}(t)$.

Observation 4.2 *A skeletal edge is enclosed at time t if and only if it is part of an edge-simple cycle in $(V, F_1 \cup F(t))$.*

Proof: First we prove the *if* part. In step 8, we collapse any cycle formed by a newly chosen edge, and we merge the clusters involved. It follows by induction on the iteration count t that all the nodes in any edge-simple cycle in $(V, F_1 \cup F(t))$ are in the same cluster in $\mathcal{C}(t)$. If a skeletal edge is part of such a cycle, both its endpoints are in the same cluster, and hence the edge is enclosed.

Now, we prove the *only-if* part. Suppose a skeletal edge s is enclosed for the first time during the course of the algorithm at time t'. Then a collapse must have occurred at time t' and the edge s participated in

the collapse. We now identify an edge-simple cycle in $(V, F_1 \cup F(t'))$ containing s. Consider the cycle that collapsed at time t'. Each cluster in the collapsing cycle is one-connected using edges in $F_1 \cup F(t'-1)$ by Proposition 4.1. So we can form a cycle containing s using the edges in this collapsing cycle, along with the paths within each cluster in this cycle in the graph $(V, F_1 \cup F(t'))$ □

Note that a skeletal edge, once enclosed, stays enclosed for the rest of the algorithm.

Observation 4.3 *Let N be any connected component in (V, F_1). Then the set of nodes in N that occur in the same cluster in $\mathcal{C}(t)$ are two-connected in $(V, F_1 \cup F(t))$.*

Proof: Using induction on the iteration count t, we can show that the set of edges in N with both endpoints in the same cluster in $\mathcal{C}(t)$ form a subtree of N. Each edge in this subtree is thus enclosed at time t by definition and hence its endpoints are two-connected by Observation 4.2. Since two-connectivity is a transitive relation on the nodes, all the nodes in this subtree are two-connected. □

Definition: For any iteration number t of the algorithm, we define $F_1(t)$ and $F_2(t)$ to be the subset of edges in F_1 and F_2 respectively with their endpoints in different clusters in $\mathcal{C}(t)$. Note that $F_2(t)$ is just the set of edges in F_2 chosen after iteration t.

From the running of the main algorithm, we have the following lemma.

Observation 4.4 *Each connected component of F_1 is completely contained in some cluster in $\mathcal{C}(t_f)$.*

Proof: Consider the graph $(\mathcal{C}(t_f), F_1(t_f))$ that F_1 induces on the set of clusters at the time of termination of the algorithm. We claim that $F_1(t_f)$ is empty. Otherwise, consider a nonsingleton connected component H of $(\mathcal{C}(t_f), F_1(t_f))$. Each cluster in H has degree at least one. By definition of the activity function f', since each cluster is inactive at time t_f, each cluster in H has degree at least two. Hence H contains a cycle. But the algorithm collapses any such cycle into a single cluster and thus this is impossible. □

The observation above implies that every skeletal edge is enclosed at time t_f. Using Observation 4.2, this implies that each tree in the initial skeletal forest (V, F_1) is two-connected in the graph $(V, F_1 \cup F)$. By the definition of clean-up, we can prove the following proposition by backwards induction on the iteration count t.

Proposition 4.5 *The graph $(V, F_1 \cup F_2(t) \cup F(t))$ does not contain any skeletal bridge edge.*

Since $F_2(0) = F_2$, the above proposition implies that there is no skeletal bridge edge in the final retained solution $(V, F_1 \cup F_2)$. Thus we have the following lemma.

Lemma 4.6 *Each component N of F_1 is two-connected in the graph $(V, F_1 \cup F_2)$.*

This proves the first part of Theorem 3.5.

Definition: Note that the edges in $F_1(t)$ and $F_2(t)$ defined earlier have their endpoints in different clusters in $\mathcal{C}(t)$. So we can define an auxiliary graph $G(t)$ on the node set $\mathcal{C}(t)$ by $G(t) = (\mathcal{C}(t), F_1(t) \cup F_2(t))$. For each edge $g = (a, b)$ in $F_1(t) \cup F_2(t)$, we have an edge (A, B) in the graph $G(t)$ such that $A \ni a$ and $B \ni b$. The graph $G(t)$ may be thought of as a contracted version of the solution subgraph $(V, F_1 \cup F_2)$ at time t in which all the nodes in the same cluster in $\mathcal{C}(t)$ have been contracted into a single node and only edges between clusters are retained.

Observation 4.7 *The graph $(\mathcal{C}(t), F_2(t))$ is acyclic.*

Proof: Connectivity between clusters in $\mathcal{C}(t)$ using edges of $F_2(t)$ implies that these clusters are eventually merged into a single cluster in the course of the algorithm. Since no edges are ever chosen between nodes in the same cluster, the graph $(\mathcal{C}(t), F_2(t))$ is acyclic. □

A property of clean-up

As a result of the clean-up, we have the following lemma.

Lemma 4.8 *Suppose the edge $e_t \in F_2$ is retained in F_2 because it left a skeletal edge b_t as a bridge edge in step 3 of the clean-up procedure. Then b_t is also a bridge edge in the graph $G(t') - \{e_t\}$ for any $t' < t$.*

Proof: By the definition of clean-up, the edge e_t is retained in F_2 because it left the skeletal edge b_t as a bridge edge in the graph $(V, F_1 \cup F_2(t) \cup F(t) - \{e_t\})$. Hence, b_t is also a bridge edge in the subgraph $(V, F_1 \cup F(t) - \{e_t\})$. From Observation 4.2, this implies that it is not enclosed at any time $t' < t$. Thus $b_t \in F_1(t')$.

Note that we can obtain the graph $G(t') - \{e_t\}$ from $(V, F_1 \cup F_2(t) \cup F(t) - \{e_t\})$ by a series of edge-contractions and edge-deletions not involving b_t. Each cluster in $\mathcal{C}(t')$ is one-connected using edges in $F_1 \cup F(t)$ by Proposition 4.1; we can therefore obtain a node in $\mathcal{C}(t')$ by contracting along the edges between nodes in V in this cluster. It remains to show that we can obtain $G(t') - \{e_t\}$ from the contracted graph by deleting edges. Now by definition, $F_1(t') \subseteq F_1$. Also, since $t' < t$, the set of edges in F_2 chosen after time t' is a subset of all the edges chosen until time t and the set of edges in F_2 chosen after time t, i.e., $F_2(t') \subseteq F(t) \cup F_2(t)$. We conclude that the edges in $G(t')$ are $F_1(t') \cup F_2(t')$, a subset of $F_1 \cup F_2(t) \cup F(t)$, so we can obtain $G(t') - \{e_t\}$ from the contracted graph by deleting edges. Note that since $b_t \in F_1(t')$, we do not use it in any of these operations. Furthermore, neither of these operations destroys bridge edges and so b_t remains a bridge in the graph $G(t') - \{e_t\}$. □

We shall use the above lemma in the proof of the performance guarantee.

5 The Performance Guarantee

Our analysis of the algorithm is modeled on that of [6]. In this section, we prove Theorem 3.6. From the construction of the $\tilde{y}$ values and the choice of edges in F, we get the following lemma directly.

Lemma 5.1 *The solution $\tilde{y}$ satisfies $\tilde{y}_S > 0$ only when $|\Gamma(S) \cap F_1| = 1$. Furthermore, it obeys the normal packing constraints on edges not in F_1. Namely, for any edge $e \in E - F_1$, we have $\sum_{S:\Gamma(S) \ni e} \tilde{y}_S \leq c_e$. Also, for any edge in F, this constraint is tight, i.e., for $e \in F$, $c_e = \sum_{S:\Gamma(S) \ni e} \tilde{y}_S$.*

The above lemma proves the second part of Theorem 3.5.

We restate Theorem 3.6 and in the remainder of this section, prove it.

Theorem 3.6 Let F_2 be the set of edges chosen by the second phase algorithm and let $\tilde{y}$ be the dual solution it implicitly constructs. Then

$$\sum_{e \in F_2} c_e \leq (2 - \frac{2}{|A|}) \sum_{S \subset V} \tilde{y}_S$$

At any time t in the running of the algorithm, recall that the set of nodes in $\mathcal{C}(t)$ can be partitioned into active nodes and dead nodes, representing clusters with f' value 1 and 0 respectively. Let $k(t)$ denote the number of active nodes in $\mathcal{C}(t)$. We prove the following theorem in the remainder of the section and use it in the induction step of our proof of Theorem 3.6.

Theorem 5.2 *For each iteration number t, the sum of the degrees of the active nodes in the graph $(\mathcal{C}(t), F_2(t))$ is at most $2(k(t) - 1)$.*

The proof of Theorem 3.6 is adapted from a proof by Goemans and Williamson [6]. They use a lemma similar to our Theorem 5.2 in order to prove Theorem 3.6 by induction on the iteration count t. However, our proof of Theorem 5.2 differs completely from the proof of its counterpart in [6]. We now show that the above theorem suffices to prove Theorem 3.6.

Proof of Theorem 3.6: The proof is by induction on the iteration count t. From Lemma 5.1, we can write $\sum_{e \in F_2} c_e = \sum_{e \in F_2} \sum_{S:\Gamma(S) \ni e} \tilde{y}_S$. By changing the order of summation we can rewrite the above double sum as follows: $\sum_{e \in F_2} c_e = \sum_S \tilde{y}_S \cdot |\{e \in \Gamma(S) : e \in F_2\}|$. Let $\tilde{y}_S(t)$ denote the value of $\tilde{y}_S$ just before the t^{th} iteration of the algorithm. We show by induction on the iteration count t that

$$\sum_S \tilde{y}_S(t) \cdot |\{e \in \Gamma(S) : e \in F_2\}| \leq (2 - \frac{2}{|A|}) \sum_{S \subset V} \tilde{y}_S(t)$$

The inequality clearly holds when $t = 0$. Let δ denote the reduced cost of the edge e_t chosen at time t. At this iteration of the algorithm, the increase in the left-hand side of the above inequality is

$$\sum_{C \in \mathcal{C}(t): f'(C)=1} \delta \cdot |\{e \in \Gamma(C) : \ e \in F_2\}|$$

For the induction step, we must prove that this increase is bounded by the increase in the right-hand side, namely by

$$(2 - \frac{2}{|A|}) \cdot \delta \cdot |C \in \mathcal{C}(t) : f'(C) = 1|$$

By definition, $k(t) = |C \in \mathcal{C}(t) : f'(C) = 1|$. In terms of the graph $(\mathcal{C}(t), F_2(t))$, we must prove that the sum of the degrees of the active nodes is at most $2(k(t) - \frac{k(t)}{|A|})$.

Each leaf in the initial forest F_1 is a terminal and the number of active nodes in $\mathcal{C}(t)$ at any time t is at most the number of leaves in F_1. Hence $k(t) \leq |A|$ at any time t. Hence it is sufficient to prove that the sum of the degrees of the active nodes in the graph $(\mathcal{C}(t), F_2(t))$ is at most $2(k(t) - 1)$. But this is exactly what Theorem 5.2 states. Thus this theorem proves the induction step. □

We shall prove Theorem 5.2 in the remainder of this section.

Proof of Theorem 5.2: The proof works by constructing a subgraph of $G(t)$ that is a forest spanning all the nodes in $G(t)$. We call this the *proof forest*; it is made up of *proof trees*. This forest will include all the edges in $F_2(t)$. It also has the property that that every edge in $F_2(t)$ is on a path between two active nodes in the forest. The following proposition asserts that we can construct such a proof forest.

Proposition 5.3 *For each iteration number t, we can construct a spanning forest of $G(t)$ that includes all the edges in $F_2(t)$ such that each such edge is on a path between two active nodes in the forest.*

Then, we *prune* the forest so as to delete each edge that does not lie on a path between two active nodes. Notice that since every edge in $F_2(t)$ is in a path between two active nodes, no edge in $F_2(t)$ gets pruned. Moreover, as a result of pruning, each leaf node in the forest will be an active node. Thus any inactive node in the forest is an internal node and has degree at least two.

Since the pruned proof forest contains every edge in $F_2(t)$, $(\mathcal{C}(t), F_2(t))$ is a subgraph of this forest. Hence the sum of the degrees of the active nodes in $(\mathcal{C}(t), F_2(t))$ is at most the sum of the degrees of the active nodes in the pruned proof forest. Let N_i denote the number of inactive nodes in the pruned proof forest. The sum of the degrees of all the nodes in the pruned forest is at most $2(k(t) + N_i - 1)$. The sum of the active node degrees is the sum of the degrees of all the nodes minus the sum of the degrees of the inactive nodes. Since each inactive node has degree at least two, the sum of the active node degrees is at most $2(k(t) - 1)$. This proves Theorem 5.2. □

Construction of the proof forest

It remains to prove Proposition 5.3. For this, we need a few definitions.

Definition: Define a node in $\mathcal{C}(t)$ to be a *skeletal node* if it has at least one edge of $F_1(t)$ incident on it. Thus the skeletal nodes are exactly the nodes that are not isolated nodes in the graph $(\mathcal{C}(t), F_1(t))$.

We say that an edge b is a *bridge edge for edge a* if b is a bridge edge in the graph $G(t) - \{a\}$.

Proof of Proposition 5.3: We prove Proposition 5.3 by proving the following lemma that asserts that we can construct the proof forest inductively. In the following lemma, t denotes the iteration count such that we are attempting to construct a proof forest for $G(t)$. For any $t' \geq t$, the set $F_2(t) - F_2(t')$ represent the set of edges in the solution subgraph that have been chosen after time t up to and including time t'.

Lemma 5.4 *Let t be an iteration number of the algorithm. Then, for every $t' \geq t$, there exists a proof forest of $(\mathcal{C}(t), F_1(t) \cup (F_2(t) - F_2(t')))$ such that the following properties hold.*

1) All the nodes in a connected component of $(\mathcal{C}(t), F_1(t))$ are in the same tree of the proof forest.

2) The proof forest contains all the edges in $F_2(t) - F_2(t')$.

3) Each edge in a nonskeletal path between two skeletal nodes in the forest is on a proof tree path between two active nodes.

4) Each skeletal edge $b \in F_1(t)$ that is not in the proof forest is a bridge edge for some nonskeletal edge in $G(t)$.

We prove Proposition 5.3 by using Lemma 5.4 with $t' = t_f$. By the second property in this lemma, we get that the proof forest includes every edge in $F_2(t)$. By Lemma 4.8, each edge e' in $F_2(t)$ is chosen at some time $t' > t$ and is part of a cycle in the graph $G(t) = (\mathcal{C}(t), F_1(t) \cup F_2(t))$. Furthermore, this cycle contains at least one skeletal edge since $(\mathcal{C}(t), F_2(t))$ is acyclic by Observation 4.7. Since the proof forest contains $F_2(t)$, this edge forms part of a nonskeletal path between skeletal nodes in the proof forest. Then the third property in Lemma 5.4 implies that this edge is on a path between two active nodes. This completes the proof of Proposition 5.3. □

Proof of Lemma 5.4: The proof is by induction on t'. Henceforth, we shall use the phrase *proof forest for t'* for any $t' \geq t$ to refer to the proof forest of the graph $(\mathcal{C}(t), F_1(t) \cup (F_2(t) - F_2(t')))$.
Basis: $(t' = t)$ In this case, the proof forest for t' is $(\mathcal{C}(t), F_1(t))$. It is easy to verify that the properties in Lemma 5.4 are true for this forest.
Induction step: Let P be the proof forest for t'. We show how to obtain a proof forest $\widehat{P}$ for $t'+1$. Let e be the edge chosen by the algorithm at $t'+1$. If e does not appear in F_2, (i.e. it was omitted during the clean-up phase), then P itself is a proof forest for $t'+1$. Therefore suppose the edge e does appear in F_2. There are two cases.

Case 1: The edge e does not form any new nonskeletal path between skeletal nodes. In this case, $P \cup \{e\}$ is acyclic, so we let the new proof forest $\widehat{P}$ be $P \cup \{e\}$. Since no edges are deleted, the first and fourth properties continue to hold. Since we added the edge e to $\widehat{P}$, the second property continues to hold. Also, since we form no new nonskeletal paths between skeletal nodes, the third property also continues to hold.

Case 2: The edge e forms a new nonskeletal path between skeletal nodes. There are two subcases.
Subcase 2a: The new nonskeletal path formed goes between two skeletal nodes in different proof trees in the forest P. Thus this path merges these two proof trees in the proof forest. In this case also, $P \cup \{e\}$ is acyclic, so we let the new proof forest $\widehat{P}$ be $P \cup \{e\}$. Since no edge is deleted, the first and fourth properties continue to hold. Since we added the edge e to $\widehat{P}$, the second property continues to hold. Also, in this case, we can inductively infer that each of the two merging trees contains an active node. Thus the edges in the newly formed nonskeletal path are on a path between two active nodes in the two merging trees. This proves the third property for the proof forest $\widehat{P}$.
Subcase 2b: The new nonskeletal path formed goes between two skeletal nodes in the same proof tree T in the forest P. In this case, addition of the edge e causes the formation of a cycle C in the proof tree T. We shall add the edge e to the proof forest P and identify a skeletal edge b in this cycle to delete so that the remaining graph is acyclic and the four properties are maintained. The first property will be maintained if we delete *any* edge in the cycle C since this still leaves all the nodes in the proof tree T connected. Since we add the edge e to form the new proof forest and do not delete any nonskeletal edge, the second property would also continue to hold. However, it is harder to find a skeletal edge to delete so as to maintain the third and fourth properties. We introduce some definition to talk about the skeletal edge we delete.

Definition: A node in the cycle C is called an *entry node* if it is reachable in $T - C$ from an active node (See Figure 2).

In the following claim, we state the properties of the skeletal edge b that we delete from C to form the new proof tree.

Claim 5.5 *There is a skeletal edge b in C such that*

1) b is on a skeletal path between two entry nodes in the cycle C.

2) b is a bridge edge for a nonskeletal edge in $G(t)$.

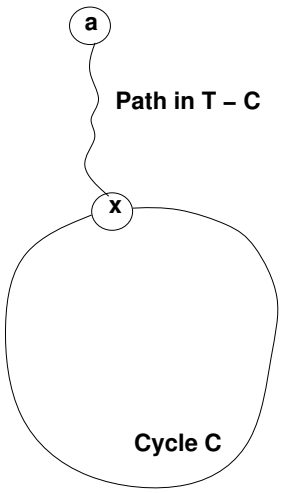

Figure 2: A node x in the cycle C is called *an entry node* if it is reachable in $T - C$ from an active node a.

Assume that the above claim holds. To form the new proof forest $\widehat{P}$ from the old, we add the edge e and delete the edge b whose existence is stated by the claim above. From the above discussion, this maintains the first two properties for $\widehat{P}$. We prove that $\widehat{P}$ also satisfies the third and fourth properties.

The fourth property follows from the second part of the above claim. To show that the third property is maintained, we must show that each edge in any nonskeletal path between skeletal nodes is on a path between two active nodes. Since the edge b is on a skeletal path between two entry nodes, each nonskeletal edge in the cycle C is on the complementary path in C between these two entry nodes. Since each of these entry nodes can reach an active node in $T - C$ by definition, all nonskeletal edges in the cycle C lie on a path between two active nodes in the new proof tree. Since the third property is inductively true for the tree T, any nonskeletal edge not in C and on a nonskeletal path between skeletal nodes is on a proof tree path between two active nodes in T. It is easy to verify that deletion of the skeletal edge b in C maintains this property for such edges. Thus the third property is maintained in the new proof tree $\widehat{P}$ in this subcase.

This completes the proof of Lemma 5.4. □

It remains to prove Claim 5.5. We do so in the remainder of this section.

Proof of Claim 5.5

We need a few preliminaries.

Claim 5.6 *For any cycle C_1 in $G(t)$, for any nonskeletal edge e' in C_1,*

(1) No edge of $G(t) - C_1$ is a bridge edge for e'.

(2) C_1 must contain a skeletal bridge edge for e'.

Proof: First we prove (1). We have by Proposition 4.5 that $G(t)$ has no skeletal bridge edge. Hence a bridge edge in $G(t) - \{e'\}$ must separate the endpoints of e'. But $C_1 - \{e'\}$ is a path between the endpoints of e', so any such bridge edge must belong to $C_1 - \{e'\}$.

Next we prove (2). By Lemma 4.8, $G(t) - \{e'\}$ must contain a skeletal bridge edge. By (1), the bridge edge must occur in C_1. □

Recall that the cycle C is the unique cycle in $T \cup \{e\}$.

Definition: A path consisting of edges of the cycle C is called a *segment* of C. A segment is *nonskeletal (skeletal)* if it consists entirely of nonskeletal (skeletal) edges. For any two nodes x and y in the cycle C, note that there are two segments between x and y. Given an edge e' of C, we can distinguish between these two segments as follows: the segment containing e' is denoted $\langle_{e'} x, y\rangle$, and the segment not containing f is denoted ${}_{e'}\langle x, y\rangle$.

Short-cuts and properties

Definition: A pair (x, y) of nodes in C is called a *short-cut* if there is a path from x to y in $G(t)$ using no edge of C.

Note that a pair (x, x) is trivially a short-cut. If $x \neq y$, we say that (x, y) is a *proper* short-cut. We have the following properties from the definition of short-cuts.

Claim 5.7 *Let e' be a nonskeletal edge of the cycle C and let (x, y) be a short-cut. Then*

(A) ${}_{e'}\langle x, y\rangle$ cannot contain a skeletal bridge edge for e'.

(B) $\langle {}_{e'}x, y\rangle$ must contain a skeletal bridge edge for e'.

Proof: Let P be the path from x to y using no edges of C. By applying Claim 5.6 to the cycle $C_1 = C \cup P - {}_{e'}\langle x, y\rangle$, we infer from (1) that ${}_{e'}\langle x, y\rangle$ contains no skeletal bridge edge for e'. This proves (A). Since C must contain such a skeletal bridge edge, it must occur in $C - {}_{e'}\langle x, y\rangle$ which is $\langle {}_{e'}x, y\rangle$. This proves (B). □

Corollary 5.8 *For each proper short-cut (x, y), the two segments between x and y each must contain a skeletal edge.*

Proof: Let S be one of the segments between x and y. Suppose for a contradiction that S consists entirely of nonskeletal edges. Let e' be one of these edges. By part (B) of Claim 5.7, $\langle {}_{e'}x, y\rangle$, which is S, contains a skeletal bridge edge for e', contradicting the fact that S is nonskeletal. □

Transition nodes and properties

Recall that we are working with a proof tree T such that C is the unique cycle in $T \cup \{e\}$.

Definition: A *transition node* is a node of C with one incident skeletal and one incident nonskeletal edge in C. Thus every maximal nonskeletal segment is bounded by transition nodes.

Lemma 5.9 *Let x be a transition node of C. Then there is an active node a_x reachable from x by a skeletal path that avoids C.*

Proof: The proof is illustrated in Figure 3. If x is itself active, we are done, so assume x is inactive. It has at least one incident skeletal edge, the one in the cycle C. Because it is inactive, it must have at least one other incident skeletal edge e' outside the cycle C. Let a_x be a leaf reachable from x by a skeletal path P_s going through e'. We show that P_s does not intersect the cycle C.

Suppose for a contradiction that the skeletal path intersects C. Let $x_1 \neq x$ be the first point of intersection of the path from x to a_x. Now x and x_1 are connected via a skeletal path P_1. They are also connected via a path P_2 along the cycle using edges of the proof tree T. Since the proof tree T is acyclic, there is a skeletal edge b' in this skeletal path P_1 from x to x_1 that is deleted in forming the proof tree. Inductively, property (4) of Lemma 5.4 is true for T and so b' must be a bridge edge for some nonskeletal edge c in $G(t)$. Since $x_1 \neq x$, note that x_1 and x are also connected in $G(t)$ via the path $P_3 = C - P_2$. Therefore the nonskeletal edge c for which b' is a bridge edge must lie in the path P_1. But since the path P_1 is skeletal, this is a contradiction. □

Observation 5.10 *For each transition node x of the cycle C, there is a entry node x' in C such that (x, x') is a short-cut.*

Proof: By Lemma 5.9, there is an active node a_x reachable from x by a skeletal path P_s that avoids C. Inductively, property (1) of Lemma 5.4 holds for T, so there is a path P_t in T from a_x to x. Let x' be the first node of P_t that lies on the cycle C; then x' is an entry node. Following P_s from x to a_x, then continuing along P_t to x' yields a path from x to x' using no edge of C. So (x, x') is a short-cut. □

Definition: For every transition node x in the cycle C, we designate an entry node x' that can be found using the above observation as the *entry node corresponding to x*.

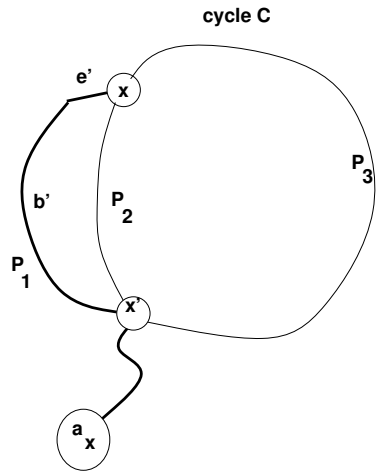

Figure 3: x is a transition node of C. e' is a skeletal edge incident to x outside C and a_x is a leaf reachable from x by a skeletal path P_s going through e'. $x_1 \neq x$ is the first node in the cycle C where the path P_s meets C. P_1, the subpath of P_s from x to x_1 is shown in dark.

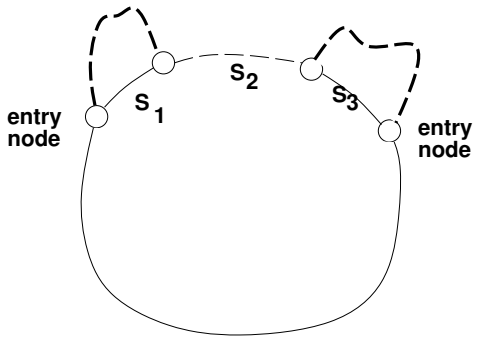

Figure 4: A segment S is said to be *forbidden* if the endpoints of S are entry nodes, and S consists of three subsegments $S_1S_2S_3$, where S_2 is a nonskeletal segment, the endpoints of S_1 form a short-cut, and the endpoints of S_3 form a short-cut. The dark dashed paths represent paths in $G(t) - C$ between the endpoints of S_1 and between the endpoints of S_3.

Forbidden segments and their properties

Definition: A segment S is said to be *forbidden* if the endpoints of S are entry nodes, and S consists of three subsegments $S_1S_2S_3$, where S_2 is a nonskeletal segment, the endpoints of S_1 form a short-cut, and the endpoints of S_3 form a short-cut (See Figure 4). For any edge f in C, if f is not in $S_1 \cup S_3$, we say that S is forbidden *for* f. This terminology is motivated by the following claim.

Let e' be any nonskeletal edge in the cycle C. By part (2) of Claim 5.6, there is a skeletal edge $b_{e'}$ in C that is a bridge edge for e'.

Claim 5.11 *Any skeletal edge $b_{e'}$ that is a bridge edge for a nonskeletal edge e' in C cannot occur in a segment that is forbidden for e'.*

Proof: Let $S = S_1S_2S_3$ be a segment forbidden for e'. Since S_2 is nonskeletal, the edge $b_{e'}$ cannot lie in this subsegment. Let the endpoints of S_1 be x and y. By definition, the edge f does not occur in S_1, so we have $S_1 = {}_{e'}\langle x, y\rangle$. Since (x, y) is a short-cut, by part (A) of Claim 5.7, ${}_{e'}\langle x, y\rangle$ cannot contain a skeletal bridge edge for e'. Hence S_1 cannot contain $b_{e'}$. Similarly, we show that S_3 cannot contain $b_{e'}$. Thus any skeletal edge $b_{e'}$ that is a bridge edge for e' cannot occur in S. □

From the definition of segments forbidden for an edge $f \in C$, we have the following observation.

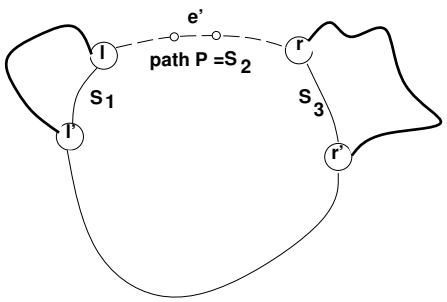

Figure 5: Lemma 5.13: e' is some nonskeletal edge in C, and P is the maximal nonskeletal path containing e' in C. Let l and r be the endpoints of P, and let their corresponding entry nodes be l' and r'. Then the segment $\langle_{e'} l', r'\rangle$ is a forbidden segment for e'. The dark lines show paths in $G(t) - C$ between each of l and r and the entry nodes l' and r' corresponding to them.

Observation 5.12 *Let S be a forbidden segment. For any nonskeletal edge f' in $C - S$, the segment S is forbidden for f'.*

The following lemma is illustrated in Figure 5.

Lemma 5.13 *Let e' be any nonskeletal edge in C, and let P be the maximal nonskeletal segment containing e'. Let l and r be the endpoints of P, and let their corresponding entry nodes be l' and r'. Then the segment $\langle_{e'} l', r'\rangle$ is a forbidden segment for e'.*

Proof: The entry nodes l' and r' cannot be internal nodes in the path P, else the short-cuts (l, l') and (r, r') would violate Corollary 5.8. Hence ${}_{e'}\langle l', l\rangle, P$, and ${}_{e'}\langle r, r'\rangle$ are contiguous edge-disjoint segments, and $\langle_{e'} l', r'\rangle = {}_{e'}\langle l', l\rangle \cup P \cup_{e'} \langle r, r'\rangle$. Thus $S = \langle_{e'} l', r'\rangle$ has entry nodes as endpoints and can be written as $S_1 S_2 S_3$ where $S_1 = {}_{e'}\langle l', l\rangle$, $S_2 = P$, and $S_3 = {}_{e'}\langle r, r'\rangle$ and (l, l') and (r, r') are short-cuts. So S is a forbidden segment. Moreover, since $e' \notin S_1 \cup S_3$, S is forbidden for a. □

Finding the skeletal edge to delete

Now we prove Claim 5.5 by providing an iterative procedure to find a skeletal edge b with the properties stated in the Claim. Note that we do not need to implement this algorithm since it is purely for the purpose of proof.

The algorithm to find the skeletal edge b maintains a nonskeletal edge f in the cycle C and a set of segments of the cycle C that are forbidden for f. On each iteration of this procedure, it either finds a skeletal edge b as required in Claim 5.5, or finds a new nonskeletal edge f' and adds a new forbidden segment to the set W. This new forbidden segment contains at least one edge not contained in any previous forbidden segment, so the number of iterations of this procedure is bounded by the size of the cycle C.

Find-skeletal-edge(f,W)

1 Initialize the nonskeletal edge $f := e$. By Lemma 5.13, we can find a segment $\langle_e u', v'\rangle$ that is forbidden for e. Initialize $W := \{\langle_e u', v'\rangle\}$.

2 Repeat

3 By part (2) of Claim 5.6, since f is a nonskeletal edge in C, there is a skeletal edge $b_f \in C$ that is a bridge edge for f. By Claim 5.11, this skeletal bridge edge b_f cannot occur in any segment in W for these are forbidden for f. So b_f must occur in a maximal segment S of C that is not overlapped by any segment in W. Note that the endpoints of S must be entry nodes.

4 If the segment S is wholly skeletal, the bridge edge b_f for f is one of the edges in this segment. This skeletal edge b_f has the properties required in Claim 5.5, so we are done in this case. We stop and output the edge b_f as the required skeletal edge.

5 Otherwise, the segment S contains at least one nonskeletal edge f'. In this case, observe that since f' is not in any of the segments in W, by Observation 5.12 each segment in W is forbidden for f'. Now we use Lemma 5.13 to find a segment $\langle_{f'} l', r'\rangle$ forbidden for f' and add this segment to the set W. We then set $f := f'$ and continue.

Notice that the set W with the segment $\langle_{f'} l', r'\rangle$ added consists of a set of segments forbidden for the new nonskeletal edge f'. Also, since f' was not in any of the forbidden segments in W earlier, but is included in the segment $\langle_{f'} l', r'\rangle$ added to W, the number of edges in the union of all the segments in W has increased by at least one.

Thus the above procedure eventually finds a skeletal edge b as required in Claim 5.5. This completes the proof of the performance guarantee of the algorithm.

6 Approximating augmentation

In this section, we restate Theorem 1.2 and prove it.

Theorem 1.2 *Given a graph $G = (V, E)$ along with an initial subgraph G_0 with edge set E_0, the* minimum-cost augmentation problem *is to choose a minimum-cost set of edges in G whose addition to G_0 yields a graph that obeys the constraints of (IP). This problem can be approximated in $O(n^2 \log n)$ time within a factor of $2(1-\frac{1}{|A|})$ if the incidence vector, x_0 of the initial subgraph G_0 obeys the one-connectivity constraints*

$$x_0(\Gamma(S)) \geq f(S) \quad \forall S \ : \ \emptyset \neq S \subset V$$

Otherwise the performance guarantee is $3(1 - \frac{1}{|A|})$.

Proof: First we prove the weaker bound. For this, we modify the cost function c on the edges of G by setting the cost of the edges in the initial subgraph G_0 to zero. Let c' be the modified cost function. Then we apply the algorithm in Theorem 1.1 to the graph G with cost function c'. The algorithm finds a set of edges F' whose incidence vector is feasible for (IP). Moreover, the cost of the set of edges F' is at most $3(1 - \frac{1}{|A|})$ times the cost of an optimum solution to (IP) under the cost function c'. Consider the minimum-cost augmentation of G_0. Let the cost of the edges in this augmentation be Aug^*. This set of edges along with the zero-cost edges in E_0 is a feasible solution to (IP) of value Aug^* under c'. Thus the cost of the edges in F' is at most $3(1 - \frac{1}{|A|})Aug^*$ under c'. Finally we observe that since F' is feasible for (IP), the edges in $F' - E_0$ form a valid augmentation of E_0. This augmentation has cost at most $3(1 - \frac{1}{|A|})Aug^*$ under c. This proves the weaker bound.

We now turn to the case when the incidence vector of the edges in E_0 satisfies the one-connectivity constraints:

$$x(\Gamma(S)) \geq f(S) \quad \forall S \ : \ \emptyset \neq S \subset V$$

In this case, we choose an arbitrary spanning forest F of G_0. It is easy to verify that the incidence vector of F satisfies the one-connectivity constraints. We then define a *minimal* version F_1 of F by retaining only the edges in F that are critical. Formally, we define

$$F_1 \leftarrow \{e \in F \ : \text{ for some connected component } N \text{ of } (V, F - \{e\}),\ f(N) = 1\}$$

It follows from the results in [6] that if F is feasible for the one-connectivity constraints, then so is F_1. We can then use F_1 as the skeletal forest and apply the second-phase algorithm in Section 4. However, as before, we first modify the cost function c on the edges of G by setting to zero the cost of the edges in E_0. We apply the algorithm in Section 4 to the graph G with the modified cost function c' and the input skeletal forest F_1. The algorithm chooses a set of edges F_2 disjoint from F_1. We first show that $F_2 - E_0$ provides a valid augmentation and then we show that it has low cost.

By Lemma 3.9, the subgraph $F_1 \cup F_2$ is feasible for (IP). Since F_1 is contained in E_0, the subgraph $F_2 - E_0$ is a valid augmentation of E_0. Now we show that $F_2 - E_0$ has low cost under c. It suffices to show that F_2 has low cost under c'. By Lemma 3.8, the cost of the edges in F_2 is at most $2(1 - \frac{1}{|A|})$ times that of an optimum solution to (IP) under the cost function c'. The optimum augmentation of G_0 together with E_0 is a feasible solution to (IP). The cost of this solution under c' is just the cost of

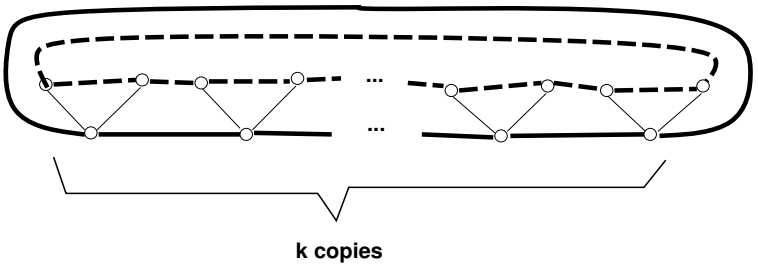

Figure 6: An example showing that the performance bounds of the algorithms in Theorems 1.1 and 1.2 are existentially tight.

the augmentation. Hence the cost of the edges in F_2 is at most $2(1 - \frac{1}{|A|})$ times the cost of the optimal augmentation. This completes the proof of the stronger bound in Theorem 1.2. □

7 Implementation issues

We note that our algorithm can be implemented in time $O(n^2 \log n)$ where $n = |V|$ using the techniques in [6]. We address the implementation of the clean-up steps. The important observation here is the initial forest F_1 and the candidate set of edges F both form acyclic subgraphs. Hence the total number edges in them is $O(n)$. At each step of clean-up, we identify the two-connected components of a graph with $O(n)$ edges. The time taken is linear in the number of edges and hence the time to implement each step of clean-up is linear in n. Since there are only a linear number of clean-up steps, the total time for performing the clean-up is $O(n^2)$. The overall running time is thus dominated by the time for implementing the main loop and is $O(n^2 \log n)$.

8 Conclusions and open problems

We have described approximation techniques for a variety of two-connected subgraph problems. We also extended these results to derive approximation algorithms for augmentation problems in this setting.

We note that the performance bounds of the algorithms we have presented in Theorems 1.1 and 1.2 are existentially tight. This can be seen by their performance on an example graph taken from [3]. This graph is reproduced in Figure 6. The thick (solid and dashed) edges are of unit cost and the thin edges have cost ϵ close to zero. We use the proper function f defined by $f(S) = 1$ for every nonempty proper subset S of V, i.e, we seek a minimum-cost two-connected subgraph. The initial spanning tree constructed in the first phase of our algorithm may consist of all the thick solid lines but one and all the thin lines of cost $(k-1) + 2k\epsilon$. All but one of the thick dashed lines may be chosen as the second phase solution. The cost of this set is $(2k-1)$. Thus our algorithm outputs a solution of total cost $3k-2+2k\epsilon$. A minimum-cost two-connected subgraph of this graph consists of alternating thick dashed lines with all the thin lines. This subgraph has cost $k + 2k\epsilon$. Thus this example also shows that the performance bounds of our algorithms are existentially tight.

It remains an important open problem to design approximation algorithms with a better performance guarantee for this problem. It would also be interesting to investigate if the techniques in this paper extend to generalized k-connected network design problems.

References

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

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

[3] 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.

[4] Greg N. Fredrickson, and Joseph Ja'Ja', "On the relationship between the biconnectivity augmentation and Travelling Salesman problems", *Theoretical Computer Science 19* (1982), pp. 189-201.

[5] M. X. Goemans, and D. J. Bertsimas, "Survivable Networks, Linear Programming Relaxations and the Parsimonious Property", OR 216-90, Center for Operations Research, MIT (1990).

[6] M. X. Goemans, and D. P. Williamson, "A general approximation technique for constrained forest problems", *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 307-316.

[7] M. Grotschel, and C. L. Monma, "Integer polyhedra arising from certain network design problems with connectivity constraints," *SIAM J. on Discrete Math.*, Vol 3, No. 4 (1990), pp. 502-523.

[8] M. Grotschel, C. L. Monma, and M. Stoer, "Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints," to appear, *Oper. Res.,* (1992).

[9] S. Khuller, and R. Thirumella, "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.*

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

[11] C. L. Monma, and C. W. Ko, "Methods for designing survivable communication networks," *NATO Advanced Research Workshop*, Denmark, (1989).

[12] C. L. Monma, B. S. Munson, and W. R. Pulleybank, "Minimum-weight two-connected spanninf networks," *Math. Programming, 46* (1990), pp. 153-171.

[13] C. L. Monma, and D. F. Shallcross, "Methods for designing communication networks with certain two-connectivity survivability constraints", *Operations Research, 37*, pp. 531-541, (1989).

[14] R. Ravi, "Approximation algorithms for Steiner augmentations for two-connectivity", Technical Report TR-CS-92-21, Brown University, April (1992).

[15] K. Steiglitz, P. Weiner, and D. J. Kleitman, "The design of minimum-cost survivable networks," *IEEE Trans, on Circuit Theory*, CT-16, 4, pp. 455-460, (1969).