Algorithms for a Network Design Problem with Crossing Supermodular Demands

Vardges Melkonian
Department of Mathematics, Ohio University, Athens, Ohio 45701

Éva Tardos
Department of Computer Science, Cornell University, Ithaca, NY 14853

Abstract

We present approximation algorithms for a class of *directed* network design problems. The network design problem is to find a minimum cost subgraph such that for each vertex set S there are at least $f(S)$ arcs leaving the set S. In the last 10 years general techniques have been developed for designing approximation algorithms for *undirected* network design problems. Recently, Kamal Jain gave a 2-approximation algorithm for the case when the function f is weakly supermodular. There has been very little progress made on *directed* network design problems. The main techniques used for the undirected problems do not have simple extensions to the directed case.

András Frank has shown that in a special case when the function f is intersecting supermodular the problem can be solved optimally. In this paper, we use this result to get a 2-approximation algorithm for a more general case when f is crossing supermodular. We also extend Jain's techniques to directed problems. We prove that if the function f is crossing supermodular, then any basic solution of the LP relaxation of our problem contains at least one variable with value greater or equal to $1/4$. This result implies a 4-approximation algorithm for the class of directed network design problems.

Keywords: Network design problems; Crossing supermodular functions; Approximation algorithms; Linear programming; Basic solutions.

1 Introduction

We consider the following network design problem for directed networks. Given a directed graph with nonnegative costs on the arcs find a minimum cost subgraph where the number of arcs leaving set S is at least $f(S)$ for all subsets S. Formally, given a directed graph $G = (V, E)$ and a requirement function $f : 2^V \mapsto Z$, the network design problem is the following integer program:

$$\text{minimize} \sum_{e \in E} c_e x_e \tag{1}$$

subject to

$$\begin{aligned} \sum_{e \in \delta_G(S)} x_e &\geq f(S), && \text{for each } S \subseteq V, \\ x_e &\in \{0, 1\}, && \text{for each } e \in E, \end{aligned}$$

where $\delta_G(S)$ denotes the set of arcs leaving S. For simplicity of notation we will use $x(\delta_G(S))$ to denote $\sum_{e \in \delta_G(S)} x_e$.

The following are some special cases of interest. When $f(S) = k$ for all $\emptyset \neq S \subset V$ the problem is that of finding a minimum cost k-connected subgraph. The case when $f(S) = 1$ for all $\emptyset \neq S \subset V$ is known as Strong Connectivity problem. The directed Steiner tree problem is to find the minimum cost directed tree rooted at r that contains a subset of vertices $D \subseteq V$. This problem is a network design problem where $f(S) = 1$ if $r \notin S$ and $S \cap D \neq \emptyset$ and $f(S) = 0$ otherwise. All these special cases are known to be NP-complete. In fact, the directed Steiner tree problem contains the Set Cover problem as a special case, and hence no polynomial time algorithm can achieve an approximation better than $O(\log n)$ unless P=NP [13].

In the last 10 years there has been significant progress in designing approximation algorithm for *undirected* network design problems [1, 7, 15, 6, 10], the analog of this problem where the graph G is undirected. General techniques have been developed for the undirected case, e.g., primal-dual algorithms [1, 7, 15, 6]. Recently, Kamal Jain gave a 2-approximation algorithm for the *undirected* case when the function f is weakly supermodular. The algorithm is based on a new technique: Jain proved that in any basic solution to the linear programming relaxation of the problem *there is* a variable whose value is at least a half.

There has been very little progress made on *directed* network design problems. The main general technique used for approximating undirected network design problems, the primal-dual method does not have a simple extension to the directed case. Recently Melkonian and Tardos extended the primal-dual in [12] to the case of Strong Connectivity problem. The best-known approximation factor for this case is due to Frederickson and Jaja [5]. They use the idea of taking the union of minimum-cost in-branching and out-branching to obtain a 2-approximation algorithm. Khuller and Vishkin [11] used a similar idea to obtain a 2-approximation algorithm for the k-connected problem. Charikar et al. [2] gave the only nontrivial approximation algorithm for the Steiner tree problem on general directed graphs (an algorithm for the special case of directed acyclic graphs was given before by Zelikovski [16]). Their method provides an $O(n^\epsilon)$-approximation algorithm for any fixed ϵ.

In this paper, we consider the case when f is crossing supermodular, i.e., for every $A, B \subseteq V$ such that $A \cap B \neq \emptyset$ and $A \cup B \neq V$ we have that

$$f(A) + f(B) \leq f(A \cap B) + f(A \cup B). \tag{2}$$

Note that the k-connected subgraph problem is defined by the function $f(S) = k$ for all $\emptyset \neq S \subset V$, which is crossing supermodular. Hence the network design problem with a crossing supermodular requirement function f is still NP-hard. However, the function defining the directed Steiner tree problem is not crossing supermodular.

The paper contains two main results: a 2-approximation algorithm for the integer program, and a structural property of the basic solutions of the LP-relaxation.

The network design problem with a crossing supermodular requirement function f is a natural extension of the case of intersecting supermodular function considered by Frank [4], i.e., when the inequality (2) holds whenever A and B are intersecting. Frank [4] has shown that this special case can be solved optimally. However, when f is crossing supermodular the problem is NP-hard and no approximation results are known for this general case. In this paper, we combine Frank's result and the k-connected subgraph approximation of Khuller and Vishkin [11] to obtain a 2-approximation algorithm for the crossing supermodular case.

In the second part of the paper, we describe the basic solutions of the LP relaxation of our problem. Extending Jain's technique [10] to the case of directed graphs, we show that in any basic feasible solution of the LP relaxation (where we replace the $x_e \in \{0, 1\}$ constraints by $0 \leq x_e \leq 1$ for all arcs e), there is at least one variable with value at least quarter. Using this result, we can obtain a 4-approximation algorithm for the problem by iteratively rounding all variables above a quarter to 1.

An interesting special case of crossing supermodular function arises when the requirement of a subset S is merely a function of its cardinality $|S|$, i.e., $f(S) = g(|S|)$. It is easy to show that requiring f to be crossing supermodular is equivalent to requiring that g to be a convex function. This particularly includes the case when $f(S) = k$ for all sets. Another example of the crossing supermodular function is $f(S) = \max(|S|, k)$.

The rest of the paper is organized as follows. In Section 2 we give the 2-approximation algorithm. The main theorem stating that all basic solutions to the linear programming relaxation of (1) have a variable that is at least a quarter is given in Section 3. In Section 4 we sketch the 4-approximation algorithm that is based on this theorem. This part of the paper is analogous to Jain's paper [10] for the undirected case. Some computational results about the performance of our algorithms are given in Section 5, and we conclude with some remarks about open problems in Section 6.

2 The 2-approximation Algorithm

We consider the following network design problem:

$$\text{minimize} \sum_{e \in E} c_e x_e \tag{3}$$

subject to

$$\begin{aligned} \sum_{e \in \delta_G(S)} x_e \geq f(S), & \qquad \text{for each } S \in \rho, \\ x_e \in \{0,1\}, & \qquad \text{for each } e \in E, \end{aligned}$$

where $\delta_G(S)$ denotes the set of the arcs leaving S, and $f(S)$ is a crossing supermodular function on the set system $\rho \subseteq 2^V$.

Frank [4] has considered the special case of this problem when f is intersecting supermodular. He showed that in this case, the LP relaxation is totally dual integral (TDI), and gave a direct combinatorial algorithm to solve the problem. The case of crossing supermodular function is NP-hard (as it contains the k-connected subgraph problem as a special case). Here we use Frank's theorem to get a simple 2-approximation algorithm.

Let r be a vertex of G. Divide all the sets in ρ into two groups as follows:

$$\rho_1 = \{A \in \rho : r \notin A\}; \tag{4}$$

$$\rho_2 = \{A \in \rho : r \in A\}. \tag{5}$$

The idea is that using Frank's technique we can solve the network design problem separately for ρ_1 and ρ_2, and then combine the solutions to get a 2-approximation for the original problem.

Lemma 1 *For the set systems ρ_1 and ρ_2, the problem (3) can be solved optimally.*

Proof: For any $S \in \rho_1$, define the requirement function as $f_1(S) = f(S)$. Then the problem for the first collection ρ_1 is

$$\text{minimize} \sum_{e \in E} c_e x_e \tag{6}$$

subject to

$$\begin{aligned} \sum_{e \in \delta_G(S)} x_e &\geq f_1(S), &\qquad& \text{for each } S \in \rho_1, \\ x_e &\in \{0,1\}, && \text{for each } e \in E, \end{aligned}$$

Note that f_1 is intersecting supermodular on the set family ρ_1. For any $S_1, S_2 \in \rho_1$, we have $r \notin S_1 \cup S_2$, and so $S_1 \cup S_2 \neq V$. This together with crossing supermodularity of f implies that the requirement function $f_1(S)$ is intersecting supermodular on ρ_1. Hence the LP relaxation of (6) is TDI, and the integer program can be solved optimally.

We will use a similar idea for ρ_2. This is not an intersecting set system. To be able to apply Frank's result we will consider the reverse of the graph G defined as $G^r = (V, E^r)$ where $E^r = \{i \to j : j \to i \in E\}$. If $e = i \to j \in E$, then for $e^r = j \to i \in E^r$ define $c_{e^r} = c_e$.

Note that the network design problem with the function $f_2(S) = f(S)$ for $S \in \rho_2$ is equivalent to the problem on the reverse graph G^r with requirement function $f_2^r(V \setminus S) = f_2(S)$ for $S \in \rho_2$. Let $\rho_2^r = \{A \subseteq V : V \setminus A \in \rho_2\}$ and $f_2^r(S) = f(V \setminus S)$ for any $S \in \rho_2^r$ (see Figure 1 for an example). Then the second subproblem is

$$\text{minimize} \sum_{e \in E^r} c_e x_e \tag{7}$$

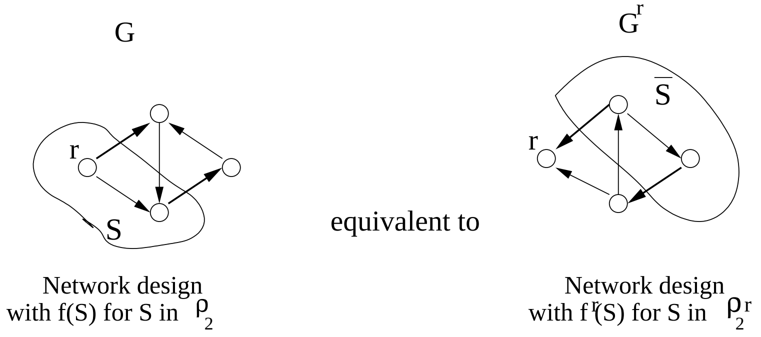


Figure 1: Example of reverse graph

subject to

$$\sum_{e \in \delta_{G^r}(S)} x_e \geq f_2^r(S), \qquad \text{for each } S \in \rho_2^r,$$
$$x_e \in \{0,1\}, \qquad \text{for each } e \in E^r,$$

Note that f_2^r is intersecting supermodular on the set system ρ_2^r: Suppose $S_1, S_2 \in \rho_2^r$ such that $S_1 \cap S_2 \neq \emptyset$. Then $\overline{S_1} \cup \overline{S_2} = \overline{S_1 \cap S_2} \neq V$, and we have that $r \in \overline{S_1} \cap \overline{S_2} \neq \emptyset$, so the sets $\overline{S_1}$ and $\overline{S_2}$ are crossing, and from the crossing supermodularity of f we get

$$\begin{aligned} f_2(S_1) + f_2(S_2) &= f(\overline{S_1}) + f(\overline{S_2}) \leq f(\overline{S_1} \cap \overline{S_2}) + f(\overline{S_1} \cup \overline{S_2}) \\ &= f(\overline{S_1 \cap S_2}) + f(\overline{S_1 \cup S_2}) = f_2(S_1 \cup S_2) + f_2(S_1 \cap S_2) \end{aligned}$$

That is, $f_2(S)$ is intersecting supermodular on ρ_2^r, and hence the linear programming relaxation of (7) is TDI, and the integer program can be solved optimally. □

We have the following simple algorithm for problem (3):

Algorithm 1 *Crossing-2approx*

1. *Solve subproblems (6) and (7) optimally.*
2. *Return the union of the optimal arc sets of the two subproblems.*

Theorem 1 *Algorithm 1 returns a solution to (3) which is within a factor of 2 of the optimal.*

Proof: Let x^* be an optimal solution to (3); $\tilde{x}$ and $\hat{x}$ be optimal solutions to (6) and (7) respectively. Since x^* is a feasible solution for both (6) and (7), we have

$$2 * \sum_{e \in E} c_e x_e^* = \sum_{e \in E} c_e x_e^* + \sum_{e \in E} c_e x_e^* \geq \sum_{e \in E} c_e \tilde{x}_e + \sum_{e \in E} c_e \hat{x}_e. \tag{8}$$

Combining the optimal arc sets of (6) and (7) we will get a solution for (3) of cost at most $\sum_{e\in E} c_e\tilde{x}_e + \sum_{e\in E} c_e\hat{x}_e$, which is within a factor of 2 of the optimal. □

3 The Structural Property of the Basic Solutions

In the previous section we gave a 2-approximation algorithm for the crossing supermodular case. In this section we give a structural property for the basic solutions of the LP relaxation of the problem. We use this description in section 4 to get a 4-approximation algorithm.

Consider the LP relaxation of the main problem (1).

$$\text{minimize} \sum_{e\in E} c_e x_e \tag{9}$$

subject to

$$\begin{aligned} x(\delta_G(S)) &\geq f(S), &\quad& \text{for each } S \subseteq V, \\ 0 \leq x_e &\leq 1, && \text{for each } e \in E. \end{aligned}$$

The second main result of the paper is the following property of the basic solutions of (9).

Theorem 2 *If $f(S)$ is crossing supermodular then in any basic solution of (9), $x_e \geq 1/4$ for at least one arc e.*

The rest of this section gives a **proof** to this theorem. The outline of the proof is analogous to Jain's [10] proof. Consider a basic solution x to the linear program.

- First we may assume without loss of generality that $x_e > 0$ for all arcs e. To see this simply delete all the arcs from the graph that have $x_e = 0$. Also assume $x_e < 1$ for all arcs e; otherwise the theorem obviously holds.
- If x is a basic solution with m non-zero variables, then there must be m linearly independent inequalities in the problem that are satisfied by equality. Each inequality corresponds to a set S, and those satisfied by equality correspond to tight sets, i.e., sets S such that $x(\delta_G(S)) = f(S)$. We use the well-known structure of tight sets to show that there are m linearly independent such equalities that correspond to a cross-free family of m tight sets. (A family of sets is cross-free if for all pairs of sets A and B in the family one of the four sets $A \setminus B$, $B \setminus A$, $A \cap B$ or the complement of $A \cup B$ is empty.)

- We will use the fact that a cross-free family of sets can be naturally represented by a forest.
- We will prove the theorem by contradiction. Assume that the statement is not true, and all variables have value $x_e < 1/4$. We consider any subgraph of the forest; using induction on the size of the subgraph we show that the k sets in this family have more than $2k$ endpoints. Applying this result for the whole forest we get that the m tight sets of the cross-free family have more than $2m$ endpoints. This is a contradiction since m arcs can have only $2m$ endpoints.

The hard part of the proof is the last step in this outline. While Jain can establish the same contradiction based on the assumption that all variables have $x_e < 1/2$, we will need the stronger assumption that $x_e < 1/4$ to reach a contradiction.

We start the proof by discussing the structure of tight sets. Call a set S *tight* if $x(\delta_G(S)) = f(S)$. Two sets A and B are called *intersecting* if $A \cap B$, $A \setminus B$, $B \setminus A$ are all nonempty. A family of sets is *laminar* if no two sets in it are intersecting, i.e., every pair of sets is either disjoint or one is contained in the other.

Two sets are called *crossing* if they are intersecting and $A \cup B$ is not the whole set V. A family of sets is *cross-free* if no two sets in the family are crossing. The key lemma for network design problems with crossing supermodular requirement function is that crossing tight sets have tight intersection and union, and the rows corresponding to these four sets in the constraint matrix are linearly dependent. Let $\mathcal{A}_G(S)$ denote the row corresponding to the set S in the constraint matrix of (1).

Lemma 2 *If two crossing sets A and B are tight then their intersection and union are also tight, and if $x_e > 0$ for all arcs then $\mathcal{A}_G(A) + \mathcal{A}_G(B) = \mathcal{A}_G(A \cup B) + \mathcal{A}_G(A \cap B)$.*

Proof: First observe that the function $x(\delta_G(S))$ is submodular, and that equation holds if and only if there are no positive weight arcs crossing from A to B or from B to A, i.e., if $x(\delta_G(A,B)) = x(\delta_G(B,A)) = 0$, where $\delta_G(A,B)$ denotes the arcs that are leaving A and entering B.

Next consider the chain of inequalities, using the crossing submodularity of $x(\delta_G(.))$, supermodularity of f and the fact that A and B are tight.

$$\begin{aligned} f(A\cup B) + f(A\cap B) &\geq f(A) + f(B) = x(\delta_G(A)) + x(\delta_G(B)) \\ &\geq x(\delta_G(A\cup B)) + x(\delta_G(A\cap B) \geq f(A\cup B) + f(A\cap B). \end{aligned}$$

This implies that both $A \cap B$ and $A \cup B$ are tight and $x(\delta_G(A,B)) = x(\delta_G(B,A)) = 0$. By our assumption that $x_e > 0$ on all arcs e, this implies that $\delta_G(A,B)$ and $\delta_G(B,A)$ are both empty, and so $\mathcal{A}_G(A) + \mathcal{A}_G(B) = \mathcal{A}_G(A \cup B) + \mathcal{A}_G(A \cap B)$. $\Box$

The main tool for the proof is a cross-free family of $|E(G)|$ linearly independent tight sets.

Lemma 3 *For any basic solution x that has $x_e > 0$ for all arcs e, there exists a cross-free family Q of tight subsets of V satisfying the following:*

- $|Q| = |E(G)|$.
- *The vectors $\mathcal{A}_G(S)$, $S \in Q$ are independent.*
- *For every set $S \in Q$, $f(S) \geq 1$.*

Proof: For any basic solution x that has $x_e > 0$ for all arcs e there must be a set of $|E(G)|$ tight sets so that the vectors $\mathcal{A}_G(S)$ corresponding to the sets S are linearly independent.

Let Q be the family of all tight sets S. We use the technique of [9] to uncross sets in Q and to create the family of tight sets in the lemma. Note that the rank of the set of vectors $\mathcal{A}_G(S)$ is $|E(G)|$. Suppose there are two crossing tight sets A and B in the family. We can delete either of A or B from our family and add both the union and the intersection. This step maintains the properties that

- all the sets in our family are tight,
- the rank of the set of vectors $\mathcal{A}_G(S)$ for $S \in Q$ is $|E(G)|$.

This is true, as the union and intersection are tight by Lemma 2, and by the same lemma the vector of the deleted element B can be expressed as a linear combination of the other three as $\mathcal{A}_G(B) = \mathcal{A}_G(A \cup B) + \mathcal{A}_G(A \cap B) - \mathcal{A}_G(A)$.

In [9] it was shown that a polynomial sequence of such steps will result in a family whose sets are cross-free. Now we can delete dependent elements to get the family of sets satisfying the first two properties stated in the lemma.

To see the last property observe that the assumption that $x_e > 0$ for all arcs e implies that all the tight sets S that have a nonzero vector $\mathcal{A}_G(S)$ must have $f(S) \geq 1$. $\Box$

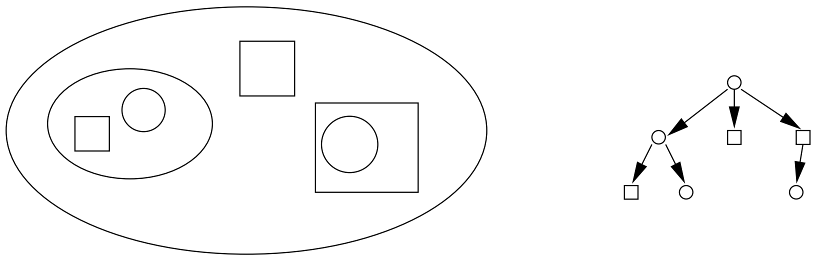


Figure 2: A laminar family with tree representation

Following the notation of András Frank we will represent a cross-free family as a laminar family with two kind of sets: round and square sets. Take any element v of $V(G)$. Let $S \in Q$ be a round set if $v \notin S$ and S be a square set if $v \in S$. It is easy to see that by representing all square sets by their complements we get a laminar family. This laminar family will provide the the natural forest representation for our cross-free family.

Lemma 4 *For any cross-free family Q, the sets $\{R \in Q$ for round sets$\}$, and the sets $\{V \setminus S$ for square sets $S \in Q\}$ together form a laminar family.*

The laminar family as given by Lemma 4 has the following *forest representation.* Let $R_1, ..., R_k$ be the round sets and $S_1, \ldots, S_l$ be the square sets of a cross-free family Q. Consider the corresponding laminar family $L = \{R_1, ..., R_k, \bar{S_1}, ..., \bar{S_l}\}$. Define the forest F as follows. The node set of F is L, and there is an arc from $U \in L$ to $W \in L$ if W is the smallest subset in L containing U. See Figure 2 for an example of a laminar family L obtained from a cross-free family Q and the tree representation of L.

Now let's give some intuition for our main proof. Lemma 3 says that $|Q| = |E(G)|$, that is,

$$\textit{There are exactly twice as many arc endpoints of } G \textit{ as subsets of } Q \qquad (10)$$

The idea of the proof comes from the following observation. Assuming that the statement of Theorem 1 is not true, that is for any $e \in E(G), x_e < 1/4$, distribute the endpoints such that each subset of Q gets at least 2 endpoints and some subsets get more than 2 endpoints. This will contradict (10). How to find this kind of distribution? The concept of incidence defined below gives the necessary connection between endpoints and subsets.

We say that an arc $e = i \to j$ of $G = (V, E)$ *leaves* a subset $U \in Q$ if $i \in U$ and $j \in \bar{U}$. Consider an arc $e = i \to j$ of $G = (V, E)$. We will define which set are the head and the tail of this arc incident to. If a node of the graph is the head or the tail of many arcs, then each

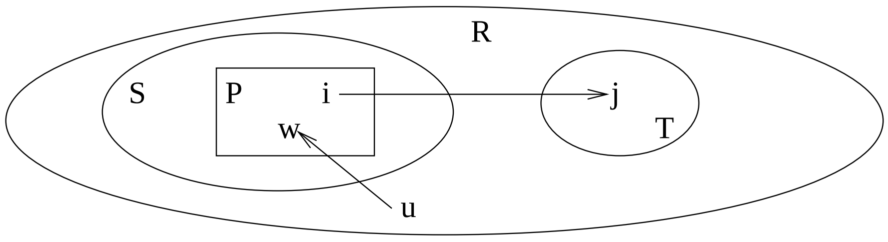


Figure 3: Incidence between endpoints and subsets

head and tail at the same node may be incident to different sets. If $U \in L$ is the smallest round subset of L such that e is leaving U then the tail of e at i is called an endpoint *incident* to U ; when no round subset satisfies the above condition then the tail of e at i is called incident to $W \in L$ if W is the smallest square subset containing both i and j. Similarly, if $U \in L$ is the smallest square subset of L such that e is leaving $\bar{U}$ then the head of e at j is called an endpoint *incident* to U ; when no square subset satisfies the above condition then the head of the arc e at j is called incident to $W \in L$ if W is the smallest round subset containing both i and j. For example, in Figure 3 the tail at i is incident to S, head at j is incident to R, the head at w is incident to P, and the tail at u is not incident to any of given subsets.

Note that by definition each arc endpoint of G is incident to at most one subset of Q. Thus, in our distribution each subset can naturally get those endpoints which are incident to it. If a subset gets more than 2 endpoints, we will reallocate the "surplus" endpoints to some other subset "lacking" endpoints so that the minimum "quota" of 2 is provided for every subset.

How to do the reallocation? Here we can use the directed forest F defined above. That is, start allocating endpoints to the subsets which correspond to the leaves of the forest. If a subset has surplus endpoints reallocate them to its parent, call this process "pumping" the surplus endpoints up the tree. If at the end of the "pumping up" process, every non-root node gets 2 endpoints and at least one root node gets more than 2 endpoints then we are getting a contradiction to (10).

Obviously, the mathematical technique to accomplish this process of pumping up is induction. That is, achieve the above allocation first for smaller subtrees and using that achieve the allocation for larger subtrees. Ideally, we would like to prove that for any rooted subtree it is possible to allocate the endpoints such that each node gets 2 endpoints and the root gets

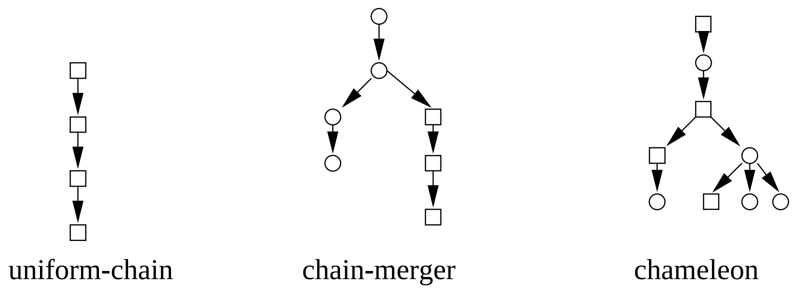


Figure 4: Tree structures

at least some specified number k which is strictly greater than 2. Unfortunately, this kind of simple statement doesn't allow us to apply the induction successfully. The point is that the roots of some tree structures can always get more than k endpoints; and if we use this loose estimate for those "surplus-rich" nodes then this might cause some of their ancestors to lack even the minimum quota of 2. That is why we need to define some tree structures and for each of them state a different k as the minimum number of endpoints that the root of that particular subtree can get.

Next we define the necessary tree structures. See Figure 4 for examples.

A *chain* is a subtree of nodes, all of which are unary except maybe the *tail* (the lowest node in the chain). Note that a node of any arity is itself a chain.

A chain with a non-unary tail is called the *survival chain* of its *head* (the highest node in the chain). So if a node is not unary then its survival chain is the node itself.

A chain of nodes having the same shape and requirement 1 is called a *uniform-chain*.

A *chain-merger* is a union of a round-shape uniform-chain and a square-shape uniform-chain such that the head of one of the chains is a child of a node of the other chain.

A subtree is called *chameleon* if

- the root has one child;
- the survival chain of the root consists of nodes of both shapes;
- the tail of the survival chain has two children.

The rest of this section is the following main lemma which completes the proof of the theorem.

Lemma 5 *Suppose every arc takes value strictly less than 1/4. Then for any rooted subtree of the forest, we can distribute the endpoints contained in it such that every vertex gets at least 2 endpoints and the root gets at least*

- *5 if the subtree is a uniform-chain;*
- *6 if the subtree is a chameleon;*
- *7 if the subtree is a chain-merger;*
- *10 if the root has at least 3 children;*
- *8 otherwise.*

Proof by induction on the height of the subtree.

First consider a *leaf* R. If the requirement of R is 1 then it is a uniform-chain and needs at least 5 endpoints allocated to it (hereafter, this allocated number will be called *label*). On the other hand, since all the variables have values less than $1/4$ then there are at least 5 arcs leaving R. Since R has no children this implies that there are at least 5 endpoints incident to it, so R gets label at least 5. If the requirement of R is greater than 1 then the same argument shows that it can get label at least 10.

For the subtrees having more than one node let's consider cases dependent on the number of children of the root R. Hereafter, without loss of generality we will assume that *R is a round node*; the other case is symmetric.

Case 1: If R has at least *four children*, by induction it gets label at least $4 * 3 = 12$ since each child has excess of at least 3.

Case 2: Suppose R has *three children*: S_1, S_2, S_3. If at least one of its children has label ≥ 6 then R will get at least $3 + 3 + 4 = 10$. Consider the case when all three children have labels 5, i.e., by induction the subtrees of all three are uniform-chains. Let's show that in this case R has an endpoint incident to it (and therefore gets label $\geq 3 + 3 + 3 + 1 = 10$). Consider cases dependent on the shapes of the children.

1. If the subtrees of S_1, S_2, S_3 are round-shaped uniform-chains then an arc leaving one of the S_i's has its head in R. Otherwise, we would have $\mathcal{A}_G(R) = \mathcal{A}_G(S_1) + \mathcal{A}_G(S_2) + \mathcal{A}_G(S_3)$ which contradicts the independence of the vectors $\mathcal{A}_G(R), \mathcal{A}_G(S_1), \mathcal{A}_G(S_2), \mathcal{A}_G(S_3)$.
2. Suppose S_1 and S_2 are round-shaped and S_3 is square-shaped. Unlike the previous case, here some arcs which start from S_1 and S_2 might enter S_3 without creating endpoints for R.

   If R has no endpoints incident to it then $f(R) < f(S_1) + f(S_2) = 1 + 1 = 2$ and hence we must have $f(R) = 1$. Thus, the sum of the values of the arcs which leave S_1 and S_2

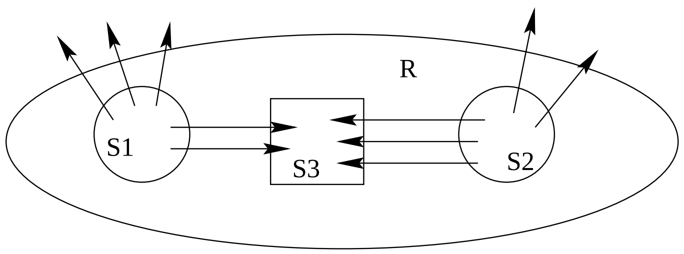


Figure 5: Case 2.2

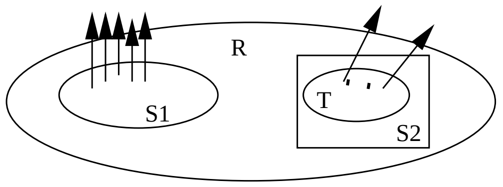
and enter S_3, is equal to $f(S_1) + f(S_2) - f(R) = 1$. Hence, the requirement of 1 of $\bar{S}_3$ is completely satisfied by this kind of arcs. In the result, $\mathcal{A}_G(R) = \mathcal{A}_G(S_1) + \mathcal{A}_G(S_2) - \mathcal{A}_G(\bar{S}_3)$ which contradicts the independence.

3. Suppose S_1 is round-shaped and S_2, S_3 are square-shaped. If there is no endpoint incident to R then all the arcs leaving S_1 should also leave R to satisfy the requirement of R which can be only 1; but this means $\mathcal{A}_G(R) = \mathcal{A}_G(S_1)$ contradicting the independence.
4. If S_1, S_2, S_3 are square-shaped then R should have endpoints to satisfy its positive requirement.

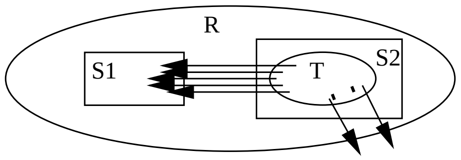
Case 3: Suppose R has *two children*, S_1 and S_2. Then R needs label 7 if its subtree is a chain-merger and label 8 otherwise. Let's consider cases in that order.

1. The subtree of R is a chain-merger if and only if the subtrees of S_1 and S_2 are uniform-chains of different shapes. In this case, by the argument used in case 2.3, R has an endpoint incident to it, so it gets label at least 3+3+1=7.
2. If none of the subtrees of S_1 and S_2 is a uniform-chain then R gets at least 4+4=8. R gets at least 8=5+3 also in the case when at least one of its children has label ≥ 7.
3. The remaining hard case is when the subtree of S_1 is a uniform-chain with label 5 and the subtree of S_2 is a chameleon with label 6. Let's show that R has an endpoint incident to it and so gets label at least 3+4+1=8. Assume that R doesn't have endpoints. Consider 2 cases:

   (a) Suppose S_1 is round-shaped. Let T be the highest round-shaped node in the survival chain of S_2 (can be S_2 itself). If an arc leaving T doesn't leave R then its head is incident to R; so all the arcs leaving T leave also R. Since $f(S_1) = 1, f(R) \in Z$ and there is no endpoint incident to R then either $f(R) = f(T) + 1$

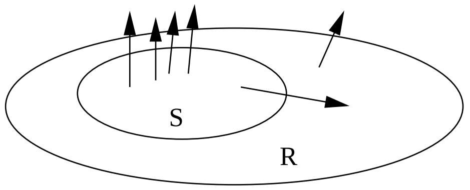


case (a) when f(R)=f(T)+1

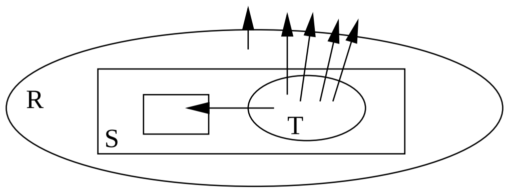


case (b)

Figure 6: Two children cases



case 4.1



case 4.2(b)

Figure 7: One child cases

or $f(R) = f(T)$. In the first case, all the arcs leaving S_1 should leave also R (see Figure 6), and $\mathcal{A}_G(R) = \mathcal{A}_G(S_1) + \mathcal{A}_G(T)$; in the second case no arc leaves both S_1 and R, and $\mathcal{A}_G(R) = \mathcal{A}_G(T)$; so the independence is violated in both cases.

(b) Suppose S_1 is square-shaped. Let T be the highest round-shaped node in the survival chain of S_2. There is an arc leaving both T and $\bar{S}_1$, otherwise we would have $\mathcal{A}_G(R) = \mathcal{A}_G(T)$. In that case, all the arcs leaving $\bar{S}_1$ should leave also T because $f(\bar{S}_1) = 1$ and $f(R) \in Z$ (see Figure 6). Thus, $\mathcal{A}_G(T) = \mathcal{A}_G(\bar{S}_1) + \mathcal{A}_G(R)$ contradicting the independence.

Case 4: Suppose R has *one child*, S. Consider cases dependent on the structure of the subtrees of R and S.

1. Suppose the subtree of R is a round-shaped *uniform-chain*, and hence R needs label 5. Then the subtree of S is also a uniform-chain. The independence of $\mathcal{A}_G(R)$ and $\mathcal{A}_G(S)$ (along with integral requirement of R) implies that R should have at least 2 endpoints incident to it (see Figure 7); thus, R gets label $\geq 3 + 2 = 5$.
2. Suppose the subtree of R is a *chameleon*, and hence R needs label 6. Consider 3 cases.

   (a) The subtree of S is a chameleon. Then the survival chain of S contains a round node T. The independence of $\mathcal{A}_G(R)$ and $\mathcal{A}_G(T)$ implies that R should have at least 2 endpoints incident to it; thus, R gets label $\geq 4+2=6$.

(b) The subtree of S, τ is a chain-merger. Then τ has a round node T such that every other round node of τ is contained in T. The independence of $\mathcal{A}_G(R)$ and $\mathcal{A}_G(T)$ implies that R should have at least 1 endpoint incident to it (see Figure 7); thus, R gets label $\geq 5+1=6$.

(c) In all other cases, S has label at least 8, so R gets at least 6.

3. Suppose the subtree of R is a *chain-merger*, and hence R needs label 7. Consider 2 cases.

   (a) The subtree of S is a chain-merger itself. Then R and S are round-shaped, and the independence of $\mathcal{A}_G(R)$ and $\mathcal{A}_G(S)$ implies that R should have at least 2 endpoints incident to it. Thus, R gets label $\geq 5+2=7$.

   (b) The subtree of S is a square-shaped uniform-chain. Since R contains only square-shaped subsets, it should have at least 5 endpoints to satisfy its requirement; thus, R gets $\geq 3+5=8$.

4. Suppose the subtree of R is *none of the 3 structures* considered above, and hence R needs label 8. Then the subtree of S can't be a uniform-chain or chameleon, and has label at least 7. Consider 2 cases.

   (a) The survival chain of S contains a round-shaped subset T. Then the independence of $\mathcal{A}_G(R)$ and $\mathcal{A}_G(T)$ implies that R should have at least 2 endpoints incident to it. Thus, R gets label $\geq 7+2=9$.

   (b) All the nodes in the survival chain of S are square-shaped. Then its tail T has at least 3 children and label ≥ 10. Based on the same independence argument, any node in the survival-chain, including S, also gets label ≥ 10. Thus, R gets at least 8. $\square$

4 The 4-approximation Algorithm via Iterative Rounding

The idea of the algorithm is to iteratively round the solutions of the linear programs derived from the main IP formulation as described below.

Based on Theorem 2, if we include the arcs with value at least a quarter in the solution of the main integer program (1) then the factor that we lose in the cost is at most 4. These arcs might not form a feasible solution for (1) yet. We reduce the LP to reflect the fact that the set of arcs is already included in the solution, and apply the method recursively.

Formally, let x^* be an optimal basic solution of (9) and $E_{1/4+}$ be the set of the arcs which take value at least 1/4 in x^*. Let $E_{res} = E - E_{1/4+}$. Consider the residual graph $G_{res} = (V, E_{res})$ and the corresponding residual LP:

$$\text{minimize} \sum_{e \in E_{res}} c_e x_e \qquad (11)$$

subject to

$$\begin{aligned} x(\delta_{G_{res}}(S)) \geq f(S) - |E_{1/4+} \cap \delta_G(S)|, &\qquad \text{for each } S \subseteq V, \\ 0 \leq x_e \leq 1, &\qquad \text{for each } e \in E_{res}. \end{aligned}$$

This residual program has the same structure as (9); the difference is that the graph G and the requirements $f(S)$ are reduced respectively to G_{res} and $f(S) - |E_{1/4+} \cap \delta_G(S)|$ considering that the arcs from $E_{1/4+}$ are already included in the integral solution. Theorem 2 can be applied to (11) if the reduced requirement function is crossing supermodular which is shown next.

A function $f : 2^V \mapsto Z$ is called submodular if $-f$ is supermodular, i.e., if for all $A, B \subseteq V$, $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$. The functions $|\delta_G(.)|$ and more generally $x(\delta_G(.))$ for any nonnegative vector x are the most classical examples of a submodular functions. The requirement function in the residual problem is the difference of a crossing supermodular function f and this submodular function, so it is also crossing supermodular.

Theorem 3 *Let $\tilde{G} = (V, \tilde{E})$ be a subgraph of the directed graph $G = (V, E)$. If $f : 2^V \mapsto Z$ is a crossing supermodular function then $f(S) - |\delta_{\tilde{G}}(S)|$ is also a crossing supermodular function.*

The high-level description of the algorithm which is based on these ideas is given in Algorithm 2.

Algorithm 2 *LP Rounding Algorithm*

1. *Find an optimal basic solution to LP (9).*

2. *Include all the arcs with values 1/4 or more in the solution of (1).*

3. *Delete all the arcs included in the solution from the graph.*

4. *Solve the residual problem recursively* **until** *no positive requirements are left.*

The algorithm requires solving the linear program (9). Note that (9) has a constraint for each subset S. Using the ellipsoid method Grötschel, Lovász and Schrijver [8] proved that such a linear program can be solved in polynomial time if there is a polynomial time separation subroutine, i.e., a method that for a given vector x can find a subset S such that $x(\delta_G(S)) < f(S)$ if such a set exists. Note that a violated set exists if and only if the minimum $\min_S(x(\delta_G(S)) - f(S))$ is negative. The function $x(\delta_G(S)) - f(S)$ is crossing submodular. Grötschel, Lovász and Schrijver [8] designed a polynomial time algorithm to minimize submodular functions. Note that a crossing submodular function is (fully) submodular if restricted to the sets $\{S : r \in S, v \notin S\}$ for any node $r \neq v$. We can obtain the minimum of a crossing submodular function by selecting a node r and computing the minimum separately over the sets $\{S : r \in S, v \notin S\}$ and $\{S : r \notin S, v \in S\}$ for all nodes $v \neq r$.

Theorem 4 *The LP Rounding Algorithm 2 returns a solution for (1) which is within a factor of 4 of the optimal.*

Proof: We prove by induction that given a basic solution x^* to (9) the method finds a feasible solution to the integer program (1) of cost at most $4cx^*$. Consider an iteration of the method. We add the arcs $E_{1/4+}$ to the integer solution. The cost of these arcs is at most 4 times the corresponding part of the fractional solution x^*, i.e., $c(E_{1/4+}) \leq 4\sum_{e\in E_{1/4+}} c_e x_e^*$. A feasible solution to the residual problem can be obtained by projecting the current solution x^* to the residual arcs. Using purification we obtain a basic solution to the residual linear program of cost at most the cost of x^* restricted to the arcs E_{res}. We recursively apply the method to find an integer solution to the residual problem of cost at most 4 times this cost, i.e., at most $4\sum_{e\in E_{res}} c_e x_e^*$. This proves the theorem. $\square$

The algorithm stated above is assuming that we solve a linear program every iteration. However, as seen by the proof, it suffices to solve the linear program (9) once, and use purification to obtain a basic solution to the residual problems in subsequent iterations.

Theorem 2 states that there is always a variable with value at least 1/4 in any basic solution of the LP relaxation. But our **conjecture** is that in any basic solution there should be a variable with value at least 1/2. And rounding this high-value variables would lead to a 2-approximation based on the same reasons given in this section for a 4-approximation. Our computational results (see Section 5) support this conjecture, at least for the Strong Connectivity Problem. We didn't have any problem instance where all variables in the LP relaxation had values strictly less than 1/2.

5 Computational experiments

We have tested our algorithms for the special case of Strong Connectivity problem when $f(S) = 1$ for all $\emptyset \neq S \subset V$. Note that for this special case, the 2-approximation algorithm of Section 2 is equivalent to the algorithm of Frederickson and Jaja [5]. The algorithms were tested on randomly generated instances.

We have used C as our implementation environment. The linear programs were solved with CPLEX 6.6.1. We used modified versions of lpex2.c and mipex2.c files of CPLEX to call the LP and MIP solvers within our main C code. All experiments were performed on a conventional Sun UltraSparc workstation.

Implementing the LP rounding algorithm

Our results obtained in Section 3 showed that there is always a variable with value at least 1/4 in any basic solution of the LP relaxation. But our conjecture is that in any basic solution there should be a variable with value at least 1/2. And rounding this high-value variables would lead to a 2-approximation; the reasons are the same as in section 4. Our computational results support this conjecture, at least for the Strong Connectivity Problem. We didn't have any problem instance where all variables had values strictly less than 1/2.

Thus, we could include only those arcs in the solution which have values 1/2 or more. Intuitively, this "1/2-rounding" version of the algorithm should return better solutions compared to the original "1/4-rounding" algorithm because we include only higher-quality arcs (arcs with high fractional values) in the solution. To make the solution quality even better we applied "sequential rounding" in our implementation: in each iteration only the arcs with the highest fractional values were included in the solution. For some instances we tested this "round-the-highest" version of the algorithm against the original "1/4-rounding" version, and "round-the-highest" delivered better results.

Results of experiments

Table 1 gives the average percentagewise cost deviations from optimum for our algorithms.

Table 1: Average percentagewise deviations from optimum

	2-approx.	LP-rounding
10 nodes	22.8	.43
15 nodes	27.6	.5
20 nodes	25.6	.52

The LP relaxation returned integer solutions for most instances. To make the statement about the performance of the LP Rounding algorithm more valid, we checked its performance only for those instances which did have fractional solutions. Table 2 gives the average deviations from optimum only for instances with fractional solutions when applying the LP Rounding algorithm; in the last column of this table we give number N_f of those instances for which the LP relaxation returns a fractional solution (and N is the total number of instances).

Table 2: Average deviations from optimum for instances with fractional solutions

	average deviations	N_f out of N
10 nodes	1.4	32 out of 100
15 nodes	.95	52 out of 100
20 nodes	.86	12 out of 20

We tested the 2-approximation Algorithm 1 also for large instances with several hundreds of nodes. Since we were not able to solve integer (or linear) programs for these large instances, the cost of the algorithm output was compared not with the optimal solution but with its lower bound. The lower bound in our case was the cost of the dual solution when the primal-dual algorithm is applied to the problem (see [12] for details on the primal-dual algorithm).

The average percentagewise deviation from the lower bound for large instances was on average 37%.

To be consistent when comparing the results for small and large networks, we have also computed the average deviations of Algorithm 1 outputs from the lower bound for small networks. The results are summarized in Table 3.

Table 3: Deviations from optimum and lower bound for Algorithm 1

	deviation from OPT	deviation from LB
10 nodes	22.8	25
15 nodes	27.6	29.6
20 nodes	25.6	26.9

As one can see from Table 3, the deviations from the lower bound are on average 7.5% bigger than the deviations from the optimum. Assuming that this ratio holds also for large networks, we could conclude that the deviations from the optimum for large instances are on average 34.5% (considering that the deviations from the lower bound are on average 37%).

Analysis of results

The LP rounding algorithm has a better performance in terms of solution quality. For most instances it delivers outputs within 1% of the optimum (see Table 1). For most instances, the LP relaxation delivered an integral solution. But even for those instances which had fractional solutions, the output of the LP Rounding algorithm was on average within 1.5% of the optimum (see Table 2).

However, the disadvantage of the LP rounding algorithm is the big size of the linear program which doesn't allow to apply the algorithm for large instances. The 2-approximation Algorithm 1 doesn't require much memory and is pretty fast (on average 6 minutes for networks with 1000 nodes). But its output is on average 25% far from the optimum for small problems, and might be about 34.5% far from the optimum for large problems.

6 Concluding remarks

Our results obtained in Section 3 showed that there is always a variable with value at least 1/4 in any basic solution of the LP relaxation. But our conjecture is that in any basic solution there should be a variable with value at least 1/2, and our computational results support this conjecture. It is an open problem to prove the conjecture theoretically.

It is interesting whether our techniques extend to other directed network design problems. In section 1 we noted that an example of the crossing supermodular function is $f(S) = \max(|S|, k)$. Note that sets S that contain almost all nodes have a very large value in this function. A more interesting function for the context of network design would be the function $f(S) = \min(|S|, k, |V \setminus S|)$. However, the minimum of convex functions is not convex, and this function is not crossing supermodular. Another interesting open problem is whether our techniques extend to the directed Steiner tree problem.

References

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

[2] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, Approximation algorithms for directed Steiner problems, Proceedings of the 9th Annual Symposium of Discrete Algorithms, 1998, pp. 192–200.

[3] J. Edmonds, Edge-disjoint branchings, In R. Rustin (editor), Combinatorial Algorithms, Academic Press, New York, 1973, pp. 91–96.

[4] A. Frank, Kernel systems of directed graphs, Acta Sci. Math, Szeged, Hungary, 41 (1979), 63–76.

[5] G. Frederickson and J. Jaja, Approximation algorithms for several graph augmentation problems, SIAM Journal of Computing 10 (1981), 270–283.

[6] M. X. Goemans, A. Goldberg, S.Plotkin, D. Shmoys, E. Tardos, and D. P. Williamson, Approximations algorithms for network design problems, Proceedings of the 5th Annual Symposium on Discrete Algorithms, 1994, pp. 223–232.

[7] M. X. Goemans and D. P. Williamson, A general approximation technique for constrained forest problem, SIAM Journal on Computing, 24 (1995), 296–317.

[8] M. Grotschel, L. Lovasz, and A. Schrijver, Geometric algorithms and combinatorial optimization, Springer-Verlag, 1988.

[9] C. A. Hurkens, L. Lovasz, A. Schrijver, and E. Tardos, How to tidy up your set-system? Proceedings of Colloquia Mathematica Societatis Janos Bolyai 52, Combinatorics, 1987, pp. 309–314.

[10] K. Jain, A factor 2 approximation algorithm for the generalized Steiner network problem, Proceedings of the 39th Annual Symposium on the Foundation of Computer Science, 1998, pp. 448–457.

[11] S. Khuller and U. Vishkin, Biconnectivity approximations and graph carvings, Journal of the Association of Computing Machinery, 41 (1994), 214–235.

[12] V. Melkonian and E. Tardos, Primal-dual-based algorithms for a directed network design problem, INFORMS Journal on Computing, to appear, http://www.math.ohiou.edu/~vardges/papers/strconn.ps

[13] R. Raz and S. Safra, A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP, Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, 1997, pp. 475–484.

[14] D. Williamson, Lecture notes on approximation algorithms, IBM Research Report RC 21409, February 1999.

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

[16] A. Zelikovski, A series of approximation algorithms for the acyclic directed Steiner tree problem, Algorithmica, 18 (1997), 99–110.