A GENERAL APPROXIMATION TECHNIQUE FOR CONSTRAINED FOREST PROBLEMS*

MICHEL X. GOEMANS† AND DAVID P. WILLIAMSON‡

Abstract. We present a general approximation technique for a large class of graph problems. Our technique mostly applies to problems of covering, at minimum cost, the vertices of a graph with trees, cycles or paths satisfying certain requirements. In particular, many basic combinatorial optimization problems fit in this framework, including the shortest path, minimum-cost spanning tree, minimum-weight perfect matching, traveling salesman and Steiner tree problems.

Our technique produces approximation algorithms that run in $O(n^2 \log n)$ time and come within a factor of 2 of optimal for most of these problems. For instance, we obtain a 2-approximation algorithm for the minimum-weight perfect matching problem under the triangle inequality. Our running time of $O(n^2 \log n)$ time compares favorably with the best strongly polynomial exact algorithms running in $O(n^3)$ time for dense graphs. A similar result is obtained for the 2-matching problem and its variants. We also derive the first approximation algorithms for many NP-complete problems, including the non-fixed point-to-point connection problem, the exact path partitioning problem and complex location-design problems. Moreover, for the prize-collecting traveling salesman or Steiner tree problems, we obtain 2-approximation algorithms, therefore improving the previously best-known performance guarantees of 2.5 and 3, respectively [4].

Key words. Approximation algorithms, combinatorial optimization, matching, Steiner tree problem, T-joins, traveling salesman problem

AMS subject classifications. 68Q25, 90C27

1. Introduction. Given a graph $G = (V, E)$, a function $f : 2^V \rightarrow \{0, 1\}$ and a non-negative cost function $c : E \rightarrow \mathbb{Q}_+$, we consider the integer program:

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

where $\delta(S)$ denotes the set of edges having exactly one endpoint in S and $x(F) = \sum_{e \in F} x_e$. The integer program (IP) can be interpreted as a very special type of covering problem in which we need to find a minimum-cost set of edges that cover all cutsets $\delta(S)$ corresponding to sets S with $f(S) = 1$. The minimal solutions to (IP) are incidence vectors of forests. We shall therefore refer to the graph problem associated with (IP) as *a constrained forest problem.* Let (LP) denote the linear programming relaxation of (IP) obtained by relaxing the integrality restriction on the variables x_e to $x_e \geq 0$. For the most part we will consider constrained forest problems corresponding to *proper* functions; that is, a function $f : 2^V \rightarrow \{0, 1\}$ such that the following properties hold:

(i) [Symmetry] $f(S) = f(V - S)$ for all $S \subseteq V$; and

* Research partially supported by DARPA contract N00014-89-J-1988.

† Department of Mathematics, MIT, Cambridge, MA 02139. Additional support provided by Air Force contract AFOSR-89-0271.

‡ School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 14853. This research was conducted while the author was a graduate student at MIT. Additional support provided by an NSF graduate fellowship.

TABLE 1
Examples of proper functions and proper constrained forest problems.

Input	$f(S)$	Minimal forests
	$f(S)=1 \quad \forall S$	Spanning trees
$s,t \in V$	$f(S)=\begin{cases} 1 & \lvert S\cap\{s,t\}\rvert = 1 \\ 0 & \text{otherwise} \end{cases}$	s-t paths
$T \subseteq V$	$f(S)=\begin{cases} 1 & \emptyset \neq S\cap T \neq T \\ 0 & \text{otherwise} \end{cases}$	Steiner trees with terminals T
$T \subseteq V$	$f(S)=\begin{cases} 1 & \lvert S\cap T\rvert \text{ odd} \\ 0 & \text{otherwise} \end{cases}$	T-joins

(ii) [Disjointness] If A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$.

We also assume that $f(V) = 0$. Many interesting families of forests can be modelled by (IP) with proper functions. In Table 1, we have indicated some examples of proper functions along with the corresponding set of minimal forests. Thus the minimum-cost spanning tree, shortest path, Steiner tree and T-join problems (see Section 3 for definitions) can be stated as *proper* constrained forest problems; that is, they can be modelled as (IP) with a proper function. Many more complex combinatorial optimization problems, such as the non-fixed point-to-point connection problem and the generalized Steiner tree problem, are also proper constrained forest problems.

Since many proper constrained forest problems are NP-complete, we focus our attention on heuristics. If a heuristic algorithm for an optimization problem delivers a solution guaranteed to be within a factor of α of optimal, it is said to have a *performance guarantee* of α. Furthermore, if it runs in polynomial time, it is called an α-approximation algorithm. In this paper, we present a $(2-\frac{2}{|A|})$-approximation algorithm for proper constrained forest problems, where $A = \{v \in V : f(\{v\}) = 1\}$. Our algorithm runs in $O(\min(n^2 \log n, mn\alpha(m,n)))$ time, where $n = |V|$, $m = |E|$, and α is the inverse Ackermann function. For the sake of the analysis, we implicitly construct a feasible solution to the dual linear program to (LP), and we prove that the value of our approximate integral primal solution is within a factor of $2-\frac{2}{|A|}$ of the value of this dual solution. Therefore, for all proper functions f, the ratio between the optimal values of (IP) and (LP) is upper bounded by $2-\frac{2}{|A|}$. This result can be contrasted with the various logarithmic upper bounds on the ratio between the optimal values of general integer covering problems and their fractional counterparts (Johnson [19], Lovász [27] and Chvátal [5]).

Our algorithm can be characterized in several ways. It is an *adaptive greedy* algorithm in which, at every iteration, the edge with minimum *reduced* cost is selected. It is adaptive in the sense that the reduced costs are updated throughout the execution of the algorithm. It can also be seen as a primal-dual algorithm in which, alternately, primal and dual updates are performed.

Our approximation algorithm generalizes many classical exact and approximate algorithms. When applied to the spanning tree problem, it reduces to Kruskal's greedy algorithm [23]. For the s-t shortest path problem, our algorithm is reminiscent of the variant of Dijkstra's algorithm that uses bi-directional search (Nicholson [28]). The algorithm is exact in these two cases. For the Steiner tree problem, we obtain the minimum spanning tree heuristic whose many variants have been described in the literature (see [39]). In the case of the generalized Steiner tree problem, our algorithm simulates Agrawal, Klein and Ravi's 2-approximation algorithm [1]. Their algorithm was instrumental in motivating our work. In particular, we generalize their use of duality from generalized Steiner trees to all proper constrained forest problems. In the process, we make their use of linear programming duality explicit and provide some conceptual simplifications since neither our algorithm nor its analysis require contractions, recursive calls to construct the forest or subdivisions of edges, as is used in the presentation of Agrawal et al. [1].

One important consequence of the algorithm is that it can be turned into a 2-approximation algorithm for the minimum-weight perfect matching problem given that the edge costs obey the triangle inequality. Our running time of $O(n^2 \log n)$ time is faster than the currently best-known algorithms that solve the problem exactly (due to Gabow [11] and Gabow and Tarjan [13]) on all but very sparse graphs. In addition, our algorithm improves upon all known approximation algorithms for this problem in either running time or performance guarantee.

Given the triangle inequality, the algorithm can also be turned into an approximation algorithm for related problems involving cycles or paths, instead of trees. This observation allows us to consider additional problems such as the traveling salesman problem, Hamiltonian location problems [25] and many other problems. Our algorithm can also be extended to handle some non-proper constrained forest problems. In general, our technique applies to many NP-complete problems arising in the design of communication networks, VLSI design and vehicle routing. We have also been able to apply the technique to the prize-collecting traveling salesman problem (given the triangle inequality) and the prize-collecting Steiner tree problem, thereby deriving the first 2-approximation algorithms for these problems.

The rest of the paper is structured as follows. In Section 2, we describe our approximation algorithm for proper constrained forest problems. We also present its analysis and an efficient implementation. In Section 3, we describe how the algorithm can be applied to the various proper constrained forest problems mentioned above. In Section 4, we show how to extend the algorithm and proof techniques to other problems, including the prize-collecting traveling salesman problem. We discuss previous work for particular constrained forest problems in Sections 3 and 4. We conclude in Section 5 with a discussion of subsequent work.

2. The Algorithm for Proper Constrained Forest Problems.

2.1. Description. The main algorithm is shown in Figure 1. The algorithm takes as input an undirected graph $G = (V, E)$, edge costs $c_e \geq 0$ for all $e \in E$, and a proper function f. The algorithm produces as output a set of edges F' whose incidence vector of edges is feasible for (IP). The basic structure of the algorithm involves maintaining a forest F of edges, which is initially empty. The edges of F will be candidates for the set of edges to be output. The algorithm loops, in every iteration selecting an edge (i, j) between two distinct connected components of F, then merging these two components by adding (i, j) to F. The loop terminates when $f(C) = 0$ for all connected components C of F; since $f(V) = 0$, the loop will finish after at most $n - 1$ iterations. The set F' of edges that are output consists of only the edges of F needed to meet the covering requirements. More precisely, if an edge e can be removed from F such that $f(C) = 0$ for all components C of $F - e$, then e is omitted from F'.

The approximation properties of the algorithm will follow from the way we choose the edge each iteration. The decision is based on a greedy construction of an implicit

Input: An undirected graph $G = (V, E)$, edge costs $c_e \geq 0$, and a proper function f
Output: A forest F' and a value LB

1 $\quad F \leftarrow \emptyset$
2 $\quad$ *Comment: Implicitly set* $y_S \leftarrow 0$ *for all* $S \subset V$
3 $\quad LB \leftarrow 0$
4 $\quad \mathcal{C} \leftarrow \{\{v\} : v \in V\}$
5 $\quad$ For each $v \in V$
6 $\quad\quad d(v) \leftarrow 0$
7 $\quad$ While $\exists C \in \mathcal{C} : f(C) = 1$
8 $\quad\quad$ Find edge $e = (i, j)$ with $i \in C_p \in \mathcal{C}$, $j \in C_q \in \mathcal{C}$, $C_p \neq C_q$ that minimizes $\epsilon = \frac{c_e - d(i) - d(j)}{f(C_p) + f(C_q)}$
9 $\quad\quad F \leftarrow F \cup \{e\}$
10 $\quad\quad$ For all $v \in C_r \in \mathcal{C}$ do $d(v) \leftarrow d(v) + \epsilon \cdot f(C_r)$
11 $\quad\quad$ *Comment: Implicitly set* $y_C \leftarrow y_C + \epsilon \cdot f(C)$ *for all* $C \in \mathcal{C}$.
12 $\quad\quad LB \leftarrow LB + \epsilon \sum_{C \in \mathcal{C}} f(C)$
13 $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$
14 $\quad F' \leftarrow \{e \in F :$ For some connected component N of $(V, F - \{e\}), f(N) = 1\}$

FIG. 1. *The main algorithm.*

solution to the dual of (LP). This dual is

$$(D) \qquad \begin{aligned} \text{Max} \quad & \sum_{S \subset V} f(S) \cdot y_S \\ \text{subject to:} \quad & \\ & \sum_{S : e \in \delta(S)} y_S \leq c_e \qquad && e \in E, \\ & y_S \geq 0 && \emptyset \neq S \subset V. \end{aligned}$$

Define an *active component* to be any component C of F for which $f(C) = 1$. In each iteration the algorithm tries to increase y_C uniformly for each active component C by a value ϵ which is as large as possible without violating the packing constraints $\sum y_S \leq c_e$. Finding such an ϵ will make a packing constraint tight for some edge (i, j) between two distinct components; the algorithm will then add (i, j) to F and merge these two components. An alternate view of this process is that the algorithm tries to find the edge (i, j) between distinct components with the minimum "reduced" cost ϵ.

We claim that the algorithm shown in Figure 1 behaves in exactly this manner. To see that the dual solution generated in steps 2 and 11 is feasible for (D), note first that initially $\sum_{e \in \delta(S)} y_S = 0 \leq c_e$ for all $e \in E$. We show by induction that the packing constraints continue to hold. Notice that it can be shown by induction that $d(i) = \sum_{S : i \in S} y_S$ for each vertex i; thus as long as vertices i and j are in different components, $\sum_{e \in \delta(S)} y_S = d(i) + d(j)$ for edge $e = (i, j)$. It follows that in a given iteration y_C can be increased by ϵ for each active component C without violating the packing constraints as long as

$$d(i) + d(j) + \epsilon \cdot f(C_p) + \epsilon \cdot f(C_q) \leq c_e,$$

for all $e = (i, j) \in E$, $i \in C_p$ and $j \in C_q$, C_p and C_q distinct. Thus the largest feasible increase in ϵ for a particular iteration is given by the formula in step 8. Once the endpoints i and j of an edge $e = (i, j)$ are in the same component, the sum

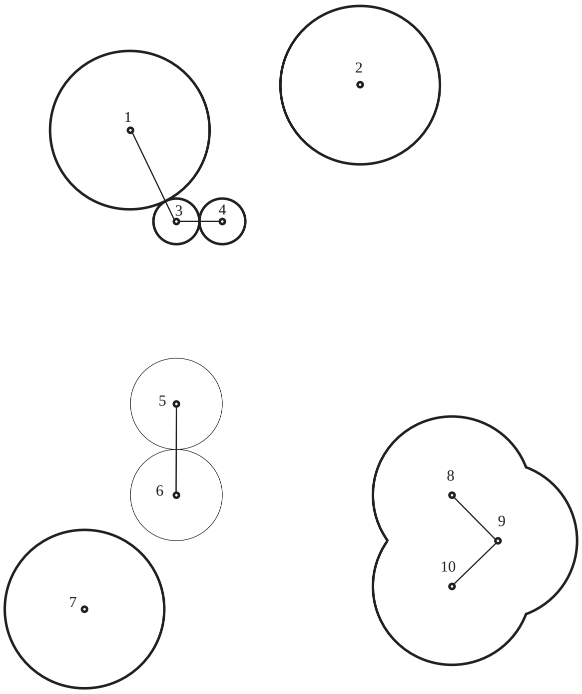

FIG. 2. *Snapshot of the algorithm.*

$\sum_{S:e\in\delta(S)} y_S$ does not increase, so that these packing constraints will continue to hold. Hence when the algorithm terminates, the dual solution y constructed by the algorithm will be feasible for (D). By the preceding discussion, we also have that $c_e = \sum_{S:e\in\delta(S)} y_S$ for each $e \in F$. Note that the value LB computed in steps 3 and 12 corresponds to the value of the dual solution y. As the value of the dual solution is a lower bound on the optimal cost, LB provides a guarantee on the performance of the algorithm for any specific instance. Furthermore, LB will also be used in the analysis below to evaluate the worst-case performance guarantee of the algorithm.

To complete our claim that the algorithm in Figure 1 behaves as described, we need to show that the edges removed in the final step of the algorithm are not necessary to meet the covering requirements; in other words, we need to show that F' is a feasible solution to (IP). We do this below.

Two snapshots of the algorithm for the proper function $f(S) \equiv |S| (\text{mod } 2)$ are shown in Figures 2 and 3. The two snapshots are one iteration apart. In both figures, the cost of an edge is the Euclidean distance between its endpoints. The radius around each vertex v represents the value $d(v)$. Thick radii represent active components, thin radii inactive components. The region of the plane defined by these radii are the so-called moats of Jünger and Pulleyblank [20, 21]. The set of edges F at the end of the main loop is shown in Figure 4, and the set of edges F' output by the algorithm is shown in Figure 5.

We can now see that the algorithm is a generalization of some classical graph algorithms. The shortest s-t path problem corresponds to the proper function $f(S) = 1$ if and only if $|S \cap \{s,t\}| = 1$. Our algorithm adds minimum-cost edges extending paths from both s and t in a manner reminiscent of Nicholson's bidirectional shortest

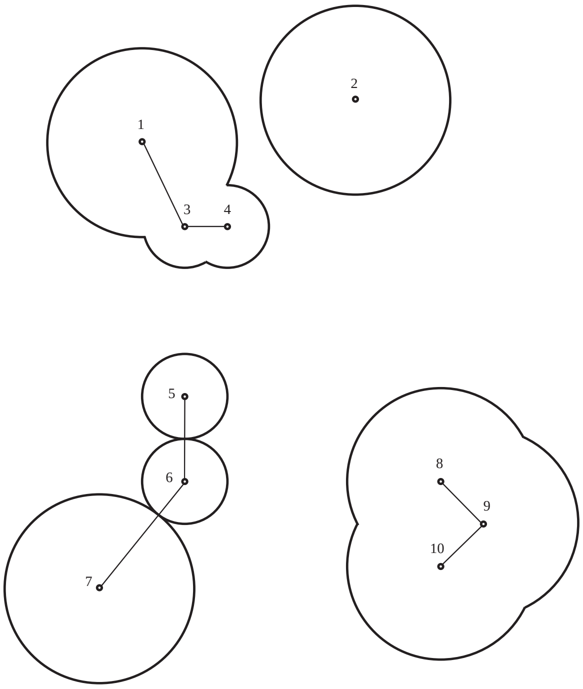

FIG. 3. *Snapshot of the algorithm one iteration later.*

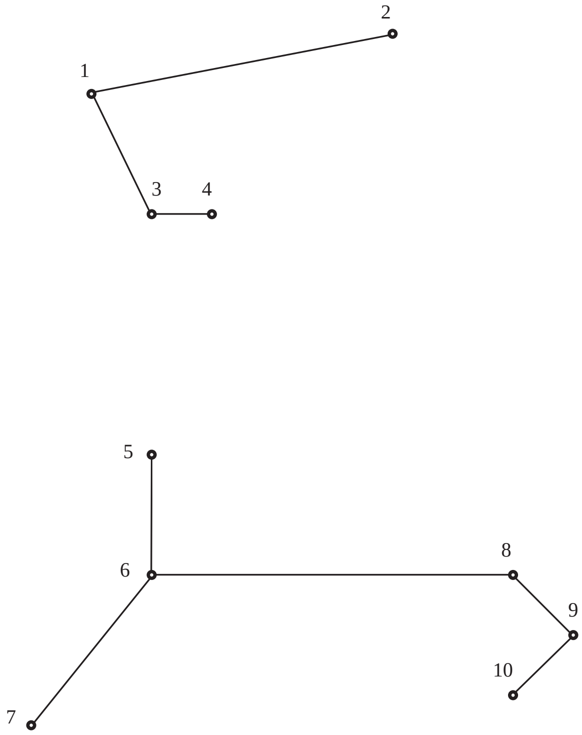

FIG. 4. *Set of edges after the main loop terminates.*

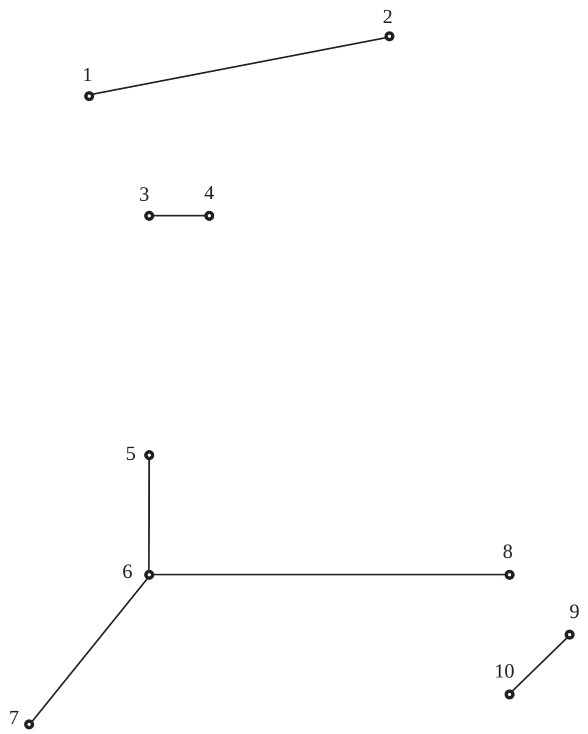

FIG. 5. *Final set of edges.*

path algorithm [28]. The main loop terminates when s and t are in the same component, and the final step of the algorithm removes all edges not on the path from s to t. Thus for this problem, whenever $y_S > 0$, $|F' \cap \delta(S)| = 1$, and whenever $e \in F'$, $\sum_{S:e\in\delta(S)} y_S = c_e$. In other words, the primal and dual feasible solutions F' and y obey the complementary slackness conditions; hence the solutions are optimal. Notice that the edge removal step is necessary to obtain a good performance guarantee in this case; this statement is also true in general. The minimum-cost spanning tree problem corresponds to a proper function $f(S) = 1$ for $\emptyset \subset S \subset V$. For this function f, our algorithm reduces to Kruskal's algorithm: all components will always be active, and thus in each iteration the minimum-cost edge joining two components will be selected. Since Kruskal's algorithm produces the optimal minimum-cost spanning tree, our algorithm will also. The solutions produced do not obey the complementary slackness conditions for (LP), but induce optimal solutions for a stronger linear programming formulation of the spanning tree problem introduced by Jünger and Pulleyblank [20].

2.2. Analysis. We now need to show that the algorithm has the properties we claim. We will begin by showing that the algorithm produces a feasible solution, and then we will turn to proving that the solution is within a factor of $(2 - \frac{2}{|A|})$ of the optimal solution. We assume throughout the ensuing discussion that F is the set of candidate edges selected by the algorithm, F' is the forest output by the algorithm, and that x' is the incidence vector of edges of F'.

OBSERVATION 2.1. *If $f(S) = 0$ and $f(B) = 0$ for some $B \subseteq S$, then $f(S-B) = 0$.*

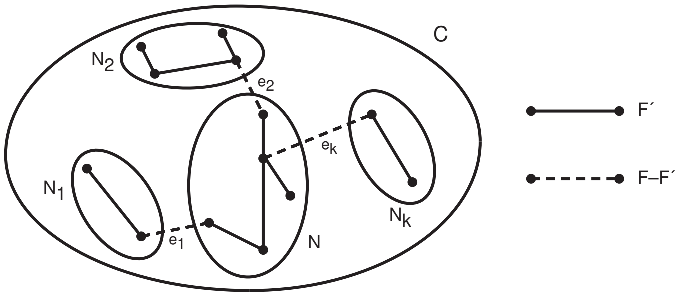

FIG. 6. *Illustration of Lemma 2.2.*

Proof. By the symmetry property of f, $f(V-S) = f(S) = 0$. By disjointness, $f((V-S)\cup B) = 0$. By symmetry again, $f(S-B) = f((V-S)\cup B) = 0$. ☐

LEMMA 2.2. *For each connected component N of F', $f(N) = 0$.*

Proof. By the construction of F', $N \subseteq C$ for some component C of F. Now, let $e_1, \ldots, e_k$ be edges of F such that $e_i \in \delta(N)$ (possibly $k = 0$). Let N_i and $C - N_i$ be the two components created by removing e_i from the edges of component C, with $N \subseteq C - N_i$ (see Figure 6). Note that since $e_i \notin F'$, it must be the case that $f(N_i) = 0$. Note also that the sets $N, N_1, N_2, \ldots, N_k$ form a partition of C. So then $f(C-N) = f(\cup_{i=1}^k N_i) = 0$ by disjointness. Because $f(C) = 0$, the observation above implies that $f(N) = 0$. ☐

THEOREM 2.3. *The incidence vector x' is a feasible solution to (IP).*

Proof. Suppose not, and assume that $x'(\delta(S)) = 0$ for some S such that $f(S) = 1$. Let $N_1, \ldots, N_p$ be the components of F'. In order for $x'(\delta(S)) = 0$, it must be the case that for all i, either $S \cap N_i = \emptyset$ or $S \cap N_i = N_i$. Thus $S = N_{i_1} \cup \ldots N_{i_k}$ for some $i_1, \ldots, i_k$. By the lemma above, however, $f(N_i) = 0$ for all i, so $f(S) = 0$ by the disjointness of f. This contradicts our assumption that $f(S) = 1$. Therefore, x' must be a feasible solution. ☐

Now we will show that the algorithm has the approximation properties that we claim. For this purpose, we use the dual solution y implicitly constructed by the algorithm. Let Z^*_{LP} be the cost of the optimal solution to (LP), and let Z^*_{IP} be the cost of the optimal solution to (IP). Obviously $Z^*_{LP} \le Z^*_{IP}$. Because y is a feasible dual solution and $y_S > 0$ only if $f(S) = 1$, it follows that $LB = \sum_{S\subset V} y_S \le Z^*_{LP}$. We will now prove the following theorem.

THEOREM 2.4. *The algorithm in Figure 1 produces a set of edges F' and a value LB such that*

$$\sum_{e\in F'} c_e \le \left(2 - \frac{2}{|A|}\right) LB = \left(2 - \frac{2}{|A|}\right) \sum_{S\subset V} y_S \le \left(2 - \frac{2}{|A|}\right) Z^*_{LP} \le \left(2 - \frac{2}{|A|}\right) Z^*_{IP}.$$

Hence the algorithm is a $(2 - \frac{2}{|A|})$-approximation algorithm for the constrained forest problem for any proper function f.

Proof. Since we know that $\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S: e\in\delta(S)} y_S$, by exchanging the summations we can rewrite $\sum_{e\in F'} c_e$ as $\sum_{S\subset V} y_S \cdot |F' \cap \delta(S)|$. To prove the theorem, we will show by induction on the main loop that

$$\sum_{S\subset V} y_S \cdot |F' \cap \delta(S)| \le \left(2 - \frac{2}{|A|}\right) \sum_{S\subset V} y_S.$$

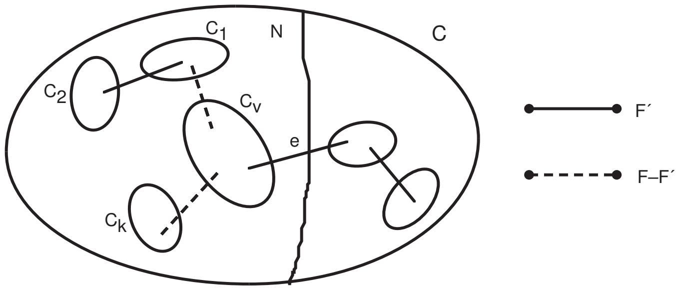

FIG. 7. *Illustration of claim that all leaves of H are active.*

Certainly the inequality holds before the first iteration of the loop, since initially all $y_S = 0$. Consider the set $\mathcal{C}$ of components at the beginning of some iteration of the loop. The left-hand side of the inequality will increase by

$$\sum_{C \in \mathcal{C}: f(C)=1} \epsilon \cdot |F' \cap \delta(C)|$$

in this iteration. If we can prove that this increase is bounded above by the increase of the right-hand side, namely

$$\left(2 - \frac{2}{|A|}\right) \epsilon \cdot |\mathcal{C}^1|,$$

where $\mathcal{C}^1 = \{C \in \mathcal{C} : f(C) = 1\}$, then we will be done.

The basic intuition behind the proof of this result is that the average degree of a vertex in a forest of at most $|A|$ vertices is at most $2 - \frac{2}{|A|}$. To begin, construct a graph H by considering the active and inactive components of this iteration as vertices of H, and the edges $e \in \delta(C) \cap F'$ for all $C \in \mathcal{C}$ as the edges of H. Remove all isolated vertices in H that correspond to inactive components. Notice that H is a forest. We claim that no leaf in H corresponds to an inactive vertex. To see this, suppose otherwise, and let v be a leaf, C_v its associated inactive component, e the edge incident to v, and C the component of F which contains C_v. Let N and $C - N$ be the two components formed by removing edge e from the edges of component C. Without loss of generality, say that $C_v \subseteq N$. The set $N - C_v$ is partitioned by some of the components of the current iteration; call these $C_1, \ldots, C_k$ (see Figure 7). Since vertex v is a leaf, no edge in F' connects C_v to any C_i. Thus by the construction of F', $f(\cup C_i) = 0$. Since $f(C_v) = 0$ also, it follows that $f(N) = 0$. We know $f(C) = 0$, so by Observation 2.1 $f(C - N) = 0$ as well, and thus by the construction of F', $e \notin F'$, which is a contradiction.

In the graph H, the degree d_v of vertex v corresponding to component C must be $|\delta(C) \cap F'|$. Let N_a be the set of vertices in H corresponding to active components, so that $|N_a| = |\mathcal{C}^1|$. Let N_i be the set of vertices in H that correspond to inactive components. Then

$$\begin{aligned}
\sum_{v \in N_a} d_v &= \sum_{v \in N_a \cup N_i} d_v - \sum_{v \in N_i} d_v \\
&\leq 2(|N_a| + |N_i| - 1) - 2|N_i| \\
&= 2|N_a| - 2.
\end{aligned}$$

This inequality holds since H is a forest with at most $|N_a|+|N_i|-1$ edges, and since each vertex corresponding to an inactive component has degree at least 2. Multiplying each side by ϵ, we obtain $\epsilon\sum_{v\in N_a} d_v \le \epsilon(2|N_a|-2)$, or

$$\epsilon \sum_{C\in\mathcal{C}^1} |F'\cap\delta(C)| \le 2\epsilon(|\mathcal{C}^1|-1) \le \left(2-\frac{2}{|A|}\right)\epsilon\cdot|\mathcal{C}^1|,$$

since the number of active components is always no more than $|A|$. Hence the theorem is proven. □

2.3. Implementing the Algorithm. We now turn to the problem of implementing the algorithm efficiently, and show how the algorithm can be made to run in $O(\min(n^2\log n, mn\alpha(m,n)))$ time. We neglect the time taken to compute f from this discussion, since we can compute $f(C)$ in $O(n)$ time for all problems of interest, and since we need to perform this computation at most $O(n)$ times.

Some of the implementation details are obvious. For example, we can maintain the components C as a union-find structure of vertices. Then all merging will take at most $O(n\alpha(n,n))$ time overall, where α is the inverse Ackermann function [33]. The two main algorithmic problems arise from selecting the edge that minimizes ϵ at each iteration, and from finding the edges in F that belong in F'. We consider each of these problems separately.

As a naive approach to finding the minimum edge, we can simply use $O(m\alpha(m,n))$ time each iteration to compute the reduced cost $\frac{c_e-d(i)-d(j)}{f(C_p)+f(C_q)}$ for each edge $e=(i,j)$ and to check whether or not the edge spans two different components. Other loop operations take $O(n)$ time, resulting in a running time of $O(mn\alpha(m,n))$ for the main loop, since there are at most $n-1$ iterations.

By being somewhat more careful, we can reduce the time taken to find the minimum edge in dense graphs to $O(n\log n)$. We need three ideas for this reduced time bound. The first idea is to introduce a notion of time into the algorithm. We let the time T be 0 at the beginning of the algorithm, and increment it by the value of ϵ each time through the main loop. The second idea is that instead of computing the reduced cost for an edge every time through the loop, we can maintain a priority queue of edges, where the key of an edge is the time T at which its reduced cost is expected to be zero. If we know whether the components of an edge's endpoints are active or inactive, and assume that the activity (or inactivity) will continue indefinitely, it is easy to compute this time T. Of course the activity of a component can change, but this occurs only when it is merged with another component, and only edges incident to the component are affected. In this case, we can recompute the key for each incident edge, delete the element with the old key, and reinsert it with the new key. The last idea we need for the lower time bound is that we only need to maintain a single edge between any two components. If there is more than one edge between any two components, one of the edges will always have a reduced cost no greater than that of the others; hence the others may be removed from consideration altogether.

Combining these ideas, we get the following algorithm for the main loop: first, we calculate the initial key value for each edge and insert each edge into the queue (in time $O(m\log n)$). Each time through the loop, we find the minimum edge (i,j) by extracting the minimum element from the queue. If $i\in C_p$ and $j\in C_q$, we delete all edges incident to C_p and C_q from the queue. For each component C_r different from C_p and C_q we update the keys of the two edges from C_p to C_r and C_q to C_r, select the one edge that has the minimum key value, then reinsert it into the queue. Since there are at most n components at any point in time, each iteration will have $O(n)$ queue insertions and deletions, yielding a time bound of $O(n \log n)$ per iteration, or $O(n^2 \log n)$ for the entire loop.

To compute F' from F, we iterate through the components C of F. Given a component C, we root the tree at some vertex, put each leaf of the tree in a separate list, and compute the f value for each of the leaves. An edge joining a vertex to its parent is discarded if the f value for the set of vertices in its subtree is 0. Whenever we have computed the f value for all the children of some vertex v, we concatenate the lists of all the children of v, add v to the list, and compute f of the vertices in the list. We continue this process until we have examined every edge in the tree. Since there are $O(n)$ edges, the process takes $O(n)$ time.

3. Applications of the Algorithm. In this section, we list several problems to which the algorithm can be applied.

The generalized Steiner tree problem. The generalized Steiner tree problem is the problem of finding a minimum-cost forest that connects all vertices in T_i for $i = 1, \ldots, p$. The generalized Steiner tree problem is a proper constrained forest problem with $f(S) = 1$ if there exists $i \in \{1, \ldots, p\}$ with $\emptyset \neq S \cap T_i \neq T_i$ and 0 otherwise. In this case, our approximation algorithm has a performance guarantee of $2 - \frac{2}{k}$, where $k = \left|\bigcup_{i=1,\ldots,p} T_i\right|$, and simulates an algorithm of Agrawal, Klein, and Ravi [1]. Their algorithm was the first approximation algorithm for this problem.

When $p = 1$, the problem reduces to the classical Steiner tree problem. For a long time, the best approximation algorithm for this problem had a performance guarantee of $(2 - \frac{2}{k})$ (for a survey, see Winter [39]) but, very recently, Zelikovsky [40] obtained an $\frac{11}{6}$–approximation algorithm. An improved $\frac{16}{9}$-approximation algorithm based upon Zelikovsky's ideas was later proposed by Berman and Ramaiyer [3].

The performance guarantee of the algorithm can be shown to be tight for this problem. When $p = 1$, our algorithm reduces to the standard minimum-cost spanning tree heuristic (see Goemans and Bertsimas [15]). The heuristic can produce solutions which have cost $2 - \frac{2}{k}$ times the optimal cost, as is shown in [15].

The T-join problem. Given an even subset T of vertices, the T-join problem consists of finding a minimum-cost set of edges that has odd degree at vertices in T and even degree at vertices not in T. Edmonds and Johnson [9] have shown that the T-join problem can be solved in polynomial time and can be formulated by the linear program (LP) with the proper function $f(S) = 1$ if $|S \cap T|$ is odd and 0 otherwise. The edge-removing step of our algorithm guarantees that the solution produced is a T-join (see below). Using our algorithm, we obtain a $(2 - \frac{2}{|T|})$-approximation algorithm for the T-join problem.

The performance guarantee of the algorithm is tight for the T-join problem. Figure 8 (a)-(c) shows an example on 8 vertices in which the minimum-cost V-join has cost $4 + 3\epsilon$, while the solution produced by the algorithm has cost 7, yielding a worst-case ratio of approximately $\frac{7}{4} = 2 - \frac{2}{8}$. Clearly the example can be extended to larger numbers of vertices and to an arbitrary set T.

When $|T| = 2$, the T-join problem reduces to the shortest path problem. Our algorithm is exact in this case, since $2 - \frac{2}{|T|} = 1$.

The minimum-weight perfect matching problem. The minimum-weight perfect matching problem is the problem of finding a minimum-cost set of non-adjacent edges that cover all vertices. This problem can be solved in polynomial time by the original primal-dual algorithm discovered by Edmonds [7]. The fastest strongly polynomial time implementation of Edmonds' algorithm is due to Gabow [11]. Its running time is $O(n(m + n \log n))$. For integral costs bounded by C, the best weakly polynomial algorithm runs in $O(m\sqrt{n\alpha(m,n)\log n}\ \log nC)$ time and is due to Gabow and Tarjan [13].

These algorithms are fairly complicated and, in fact, too time-consuming for large instances that arise in practice. This motivated the search for faster approximation algorithms. Reingold and Tarjan [30] have shown that the greedy procedure has a tight performance guarantee of $\frac{4}{3}n^{0.585}$ for general non-negative cost functions. Supowit, Plaisted and Reingold [32] and Plaisted [29] have proposed an $O(\min(n^2 \log n, m \log^2 n))$ time approximation algorithm for instances that obey the triangle inequality. Their algorithm has a tight performance guarantee of $2\log_3(1.5n)$. As shown by Gabow and Tarjan [13], an exact scaling algorithm for the maximum-weight matching problem can be used to obtain an $(1 + 1/n^a)$-approximation algorithm ($a \geq 0$) for the minimum-weight perfect matching problem. Moreover, if the original exact algorithm runs in $O(f(m,n)\log C)$ time, the resulting approximation algorithm runs in $O(m\sqrt{n\log n} + (1+a)f(m,n)\log n)$. Vaidya [34] obtains a $(3+2\epsilon)$-approximation algorithm for minimum-weight perfect matching instances satisfying the triangle inequality. His algorithm runs in $O(n^2 \log^{2.5} n \log(1/\epsilon))$ time.

The algorithm for proper constrained forest problems can be used to approximate the minimum-weight perfect matching problem when the edge costs obey the triangle inequality. We use the algorithm with the proper function $f(S)$ being the parity of $|S|$, i.e. $f(S) = 1$ if $|S|$ is odd and 0 if $|S|$ is even. This function is the same as the one used for the V-join problem. The algorithm returns a forest whose components have even size. More precisely, the forest is a V-join, and each vertex has odd degree: if a vertex has even degree, then, by a parity argument, some edge adjacent to the vertex could have been deleted so that the resulting components have even size. Thus this edge would have been deleted in the final step of the algorithm. The forest can be transformed into a perfect matching with no increase of cost by repeatedly taking two edges (u,v) and (v,w) from a vertex v of degree three or more and replacing these edges with the edge (u,w). This procedure maintains the property that the vertices have odd degree. After $O(n)$ iterations, each vertex has degree one. Since each iteration takes $O(1)$ time, the overall procedure gives an approximation algorithm for weighted perfect matching which runs in $O(n^2 \log n)$ time and has a performance guarantee of $2 - \frac{2}{n}$.

The performance guarantee of the algorithm is tight for this problem also, as in shown in Figure 8 (d).

Point-to-point connection problems. In the point-to-point connection problem, 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 a graph $G = (V, E)$ and we need to find a minimum-cost set F of edges such that each source-destination pair is connected in F [26]. This problem arises in the context of circuit switching and VLSI design. The fixed destination case in which c_i is required to be connected to d_i is a special case of the generalized Steiner tree problem where $T_i = \{c_i, d_i\}$. In the non-fixed destination case, each component of the forest F is only required to contain the same number of sources and destinations. This problem is NP-complete [26].

The non-fixed case is a proper constrained forest problem with $f(S) = 1$ if $|S \cap$

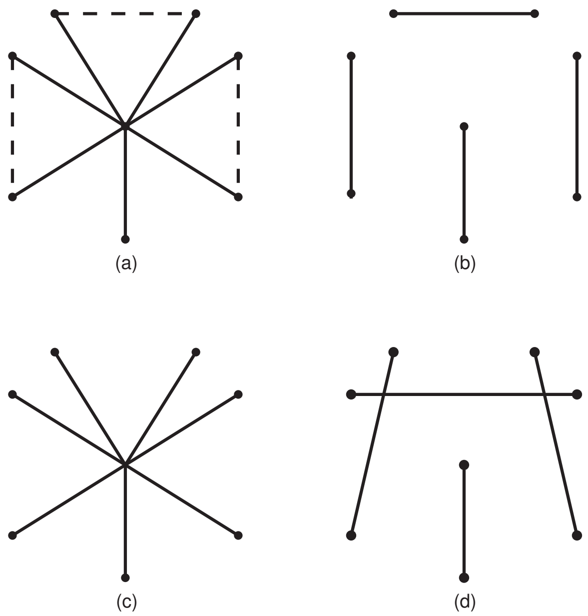

FIG. 8. *Worst-case example for V-join or matching. Graph (a) gives the instance: plain edges have cost 1, dotted edges have cost $1+\epsilon$, and all other edges have cost 2. Graph (b) is the minimum-cost solution. Graph (c) is the set of edges found by the constrained forest algorithm, and graph (d) shows a bad (but possible) shortcutting of the edges to a matching.*

$C| \neq |S \cap D|$ and 0 otherwise. For this problem, we obtain a $(2 - \frac{1}{p})$-approximation algorithm.

Exact partitioning problems. In the exact tree (cycle, path) partitioning problem, for a given k we must find a minimum-cost collection of vertex-disjoint trees (cycles, paths) of size k that cover all vertices. These problems and related NP-complete problems arise in the design of communication networks, vehicle routing and cluster analysis. These problems generalize the minimum-weight perfect matching problem (in which each component must have size exactly 2), the traveling salesman problem, the Hamiltonian path problem and the minimum-cost spanning tree problem.

We can approximate the exact tree, cycle and path partitioning problems for instances that satisfy the triangle inequality. For this purpose, we consider the proper constrained forest problem with the function $f(S) = 1$ if $S \not\equiv 0 \,(\mathrm{mod}\ k)$ and 0 otherwise. Our algorithm finds a forest in which each component has a number of vertices which is a multiple of k, and such that the cost of the forest is within $2 - \frac{2}{n}$ of the optimal such forest. Obviously the cost of the optimal such forest is a lower bound on the optimal exact tree and path partitions. Given the forest, we duplicate each edge and find a tour of each component by shortcutting the resulting Eulerian graph on each component. If we remove every kth edge of the tour, starting at some edge, the tour is partitioned into paths of k nodes each. Some choice of edges to be removed (i.e., some choice of starting edge) accounts for at least $\frac{1}{k}$ of the cost of the tour, and so we remove these edges. Thus this algorithm is a $\left(4(1 - \frac{1}{k})(1 - \frac{1}{n})\right)$-approximation algorithm for the exact tree and path partitioning problems.

To produce a solution for the exact cycle partitioning problem, we add the edge joining the endpoints of each path; given the triangle inequality, this at most doubles the cost of the solution produced. We claim, however, that the algorithm is still a $\big(4(1-\frac{1}{k})(1-\frac{1}{n})\big)$-approximation algorithm for the cycle problem. To see that this claim is true, note that the following linear program is a linear programming relaxation of the exact cycle partitioning program, given the function f above:

$$\begin{array}{lll} \text{Min} & \displaystyle\sum_{e\in E} c_e x_e & \\ \text{subject to:} & & \\ & x(\delta(S)) \geq 2f(S) & S \subset V \\ & x_e \geq 0 & e \in E. \end{array}$$

Its dual is

$$\begin{array}{lll} \text{Max} & \displaystyle 2\sum_{S\subset V} f(S)\cdot y_S & \\ \text{subject to:} & & \\ & \displaystyle\sum_{S:e\in\delta(S)} y_S \leq c_e & e \in E, \\ & y_S \geq 0 & \emptyset \neq S \subset V. \end{array}$$

We know the algorithm produces a solution y that is feasible for this dual such that $\sum_{e\in F'} c_e \leq (2-\frac{2}{n})\sum y_S$. The argument above shows how to take the set of edges F' and produce a set of edges T such that T is a solution to the exact cycle partitioning problem, and $\sum_{e\in T} c_e \leq 4(1-\frac{1}{k})\sum_{e\in F'} c_e$, so that

$$\sum_{e\in T} c_e \leq 8\left(1-\frac{1}{k}\right)\left(1-\frac{1}{n}\right)\sum y_S.$$

Since $2\sum y_S$ is the dual objective function, $2\sum y_S$ is a lower bound on the cost of the optimal exact cycle partition, Z_C^*. Thus

$$\sum_{e\in T} c_e \leq 4\left(1-\frac{1}{k}\right)\left(1-\frac{1}{n}\right) Z_C^*.$$

The proper functions corresponding to the non-fixed point-to-point connection problem, the T-join problem and the exact partitioning problems are all of the form $f(S)=1$ if $\sum_{i\in S} a_i \not\equiv 0 \pmod p$ and 0 otherwise, for some integers a_i, $i \in V$, and some integer p.

4. Extensions. The main algorithm can be extended in a number of ways to handle non-proper functions f and even other somewhat different integer programming problems. We describe these extensions in this section.

4.1. More general functions f. We can weaken the conditions for a proper function f and obtain a modified $(2-\frac{1}{n})$-approximation algorithm for the constrained forest problem. A number of new problems can be approximated under these weaker conditions; these problems are listed below. In order to solve these problems, the main algorithm must be modified to handle functions f that are not symmetric and cannot be made symmetric without violating disjointness. The central modifications that must be made involve maintaining a root vertex for each component, in order to cope with the asymmetry of f, and maintaining sets of vertices that must be connected in the final solution.

We omit discussion of the extended algorithm here because a recent algorithm of Williamson, Goemans, Mihail, and Vazirani [38] simplifies and subsumes our extended algorithm. Williamson et al. have shown how some of the results of this paper can be extended to *uncrossable* functions h. A function $h : 2^V \to \{0, 1\}$ is uncrossable if whenever $h(A) = h(B) = 1$ then either $h(A - B) = h(B - A) = 1$, or $h(A \cup B) = h(A \cap B) = 1$. Williamson et al. show that an algorithm somewhat similar to our algorithm finds solutions to (IP) for uncrossable functions that are within a factor of $2 - \frac{1}{n}$ of optimal. The algorithm can be implemented in polynomial time for many uncrossable functions, including those for which the function has the property that if $h(A) = 1$ then $h(B) = 1$ for $\emptyset \neq B \subset A$. The problems listed below fit in this category. See Williamson [36] and Goemans and Williamson [17] for a discussion of how the techniques of Williamson et al. apply to these problems.

Lower capacitated partitioning problems. The *lower capacitated partitioning problems* are like the exact partitioning problems except that each component is required to have at least k vertices rather than exactly k vertices. The lower capacitated *cycle* partitioning problem is a variant of the 2-matching problem. More precisely, the cases $k = 2$, 3 and 4 correspond to integer, binary and triangle-free binary 2-matchings respectively. The lower capacitated cycle partitioning problem is NP-complete for $k \geq 5$ (Papadimitriou [6] for $k \geq 6$ and Vornberger [35] for $k = 5$), polynomially solvable for $k = 2$ or 3 (Edmonds and Johnson [8]), while its complexity for $k = 4$ is open. Imielinska, Kalantari, and Khachiyan [18] have shown that the lower capacitated tree partitioning problem is NP-complete for $k \geq 4$.

The lower capacitated tree partitioning problem is the constrained forest problem corresponding to $f(S) = 1$ if $0 < |S| < k$ and 0 otherwise. The extended algorithm gives a $(2 - \frac{1}{n})$-approximation algorithm for this problem for any k. Furthermore, assuming the triangle inequality, this algorithm can be turned into a $(2 - \frac{1}{n})$-approximation algorithm for the lower capacitated cycle partitioning problem and a $(4 - \frac{2}{n})$-approximation algorithm for the lower capacitated path partitioning problem.

Location-design and location-routing problems. Many network design or vehicle routing problems require two levels of decisions. In the first level, the location of special vertices, such as concentrators or switches in the design of communication networks, or depots in the routing of vehicles, need to be decided. There is typically a set of possible locations and a fixed cost is associated with each of them. Once the locations of the depots are decided, the second level deals with the design or routing per se. These problems are called location-design or location-routing problems [24]. Several of these problems can be approximated using the extended algorithm. For example, we can provide a $(2 - \frac{1}{n})$-approximation algorithm for the problem in which we need to select depots among a subset D of vertices of a graph $G = (V, E)$ and cover all vertices in V with a set of cycles, each containing a selected depot [25, 24]. The goal is to minimize the sum of the fixed costs of opening our depots and the sum of the costs of the edges of our cycles. The algorithm can also be extended to the case in which every cycle is required to have at least k vertices.

4.2. Non-negative functions f. Using techniques from Goemans and Bertsimas [15], we can provide approximation algorithms for many functions $f : 2^V \to \mathbb{N}$, assuming that we can have multiple copies of an edge in the solution. Suppose f satisfies $f(S) = f(V - S)$ for all $S \subseteq V$, and, if A and B are disjoint, then

$\max\{f(A), f(B)\} \geq f(A \cup B)$. Suppose also that f assumes at most p different non-zero values, $\rho_0 = 0 < \rho_1 < \cdots < \rho_p$. Let (IP') denote the integer program (IP) with the $x_e \in \{0,1\}$ constraint replaced by the constraint $x_e \in \mathbb{N}$. Then we can show that there is an approximation algorithm for (IP') that comes within a factor of $2\sum_{k=1}^{p} \frac{\rho_k - \rho_{k-1}}{\rho_k}$ of optimal. Notice that at worst the values of f will be $0, 1, 2, 3, \ldots, f_{\max} = \max_S f(S)$, so that the performance guarantee will be at most $2\sum_{k=1}^{f_{\max}} \frac{1}{k} = O(\log f_{\max})$. The performance guarantee will also be no worse than $2p$. The algorithm for (IP') works by performing p iterations of our main algorithm. In iteration i, set $g(S) = 1$ if $f(S) \geq \rho_{p+1-i}$, $g(S) = 0$ otherwise, and call the main algorithm with function g. By the properties of f, g will be a proper function for the main algorithm. When the algorithm returns F', we make $(\rho_{p+1-i} - \rho_{p-i})$ copies of each edge, and add them to the set of edges to be output. The proof that constructing a set of edges in this way comes within a factor of $2\sum_{k=1}^{p} \frac{\rho_k - \rho_{k-1}}{\rho_k}$ of optimal is essentially the same as the proof used by Goemans and Bertsimas. We can potentially reduce the number of calls to our main algorithm by using a "scaling" technique introduced by Agrawal et al. [1] which requires $\lfloor \log f_{\max} \rfloor + 1$ iterations. In iteration i, we set $g(S) = 1$ if $f(S) \geq 2^{\lfloor \log f_{\max} \rfloor + 1 - i}$, $g(S) = 0$ otherwise, and call the main algorithm with the function g. We make $2^{\lfloor \log f_{\max} \rfloor + 1 - i}$ copies of the edges in the resulting F', and add them to the set of edges to be output. Using the Goemans and Bertsimas proof, it can be shown that this procedure results in a $(2\lfloor \log f_{\max} \rfloor + 2)$-approximation algorithm.

One application of allowing $f_{\max} > 1$ is the generalized Steiner network problem in which each pair of vertices i, j must be connected by r_{ij} edge-disjoint paths. In this case we want $f(S) = \max_{i \in S, j \notin S} r_{ij}$ For this particular problem, Agrawal et al. [1] showed how to reduce this general case to the 0-1 case.

Williamson et al. [38] have recently shown how to approximate (IP) for functions of the type mentioned above when no edge replication is allowed.

4.3. The Prize-Collecting Problems. The prize-collecting traveling salesman problem is a variation of the classical traveling salesman problem (TSP). In addition to the cost on the edges, we have also a penalty π_i on each vertex i. The goal is to find a tour on a subset of the vertices that minimizes the sum of the cost of the edges in the tour and the vertices not in the tour. We consider the version in which a prespecified root vertex r has to be in the tour; this is without loss of generality, since we can repeat the algorithm n times, setting each vertex to be the root. This version of the prize-collecting TSP is a special case of a more general problem introduced by Balas [2]. The prize-collecting Steiner tree problem is defined analogously. The standard Steiner tree problem can be seen to be a special case of the prize-collecting Steiner tree problem in which non-terminals have a penalty of zero, while terminals have a very large penalty (e.g., equal to the diameter of the graph).

Bienstock, Goemans, Simchi-Levi and Williamson [4] developed the first approximation algorithms for these problems. Their performance bounds are $5/2$ for the TSP version (assuming the triangle inequality) and 3 for the Steiner tree version. These approximation algorithms are not very efficient, however, since they are based upon the solution of a linear programming problem.

These problems do not fit in the framework of problems considered so far since they cannot be modelled by (IP). However, the main algorithm can be modified to give a $(2 - \frac{1}{n-1})$-approximation algorithm for both the prize-collecting TSP (under the triangle inequality) and the prize-collecting Steiner tree problem. Moreover, these algorithms are purely combinatorial and do not require the solution of a linear programming problem as in [4]. We will focus our attention on the prize-collecting Steiner tree problem, and at the end of the section we will show how the algorithm for the tree problem can be easily modified to yield a prize-collecting TSP algorithm.

4.3.1. The Prize-Collecting Steiner Tree. The prize-collecting Steiner tree can be formulated as the following integer program:

$$
(PC\text{-}IP) \qquad
\begin{array}{lll}
\text{Min} & \displaystyle\sum_{e\in E} c_e x_e + \sum_{T\subset V; r\notin T} z_T\big(\sum_{i\in T}\pi_i\big) & \\
\text{subject to:} & & \\
& \displaystyle x(\delta(S)) + \sum_{T\supseteq S} z_T \geq 1 & S\subset V; r\notin S \\
& \displaystyle\sum_{T\subset V; r\notin T} z_T \leq 1 & \\
& x_e \in \{0,1\} & e\in E \\
& z_T \in \{0,1\} & T\subset V; r\notin T.
\end{array}
$$

Intuitively, z_T is set to 0 for all T except the set T of all vertices not spanned by the tree of selected edges. A linear programming relaxation $(PC\text{-}LP)$ of the integer program can be created by replacing the integrality constraints with the constraints $x_e \geq 0$ and $z_T \geq 0$ and dropping the constraint $\sum_T z_T \leq 1$ (in fact, including this constraint does not affect the optimal solution). The LP relaxation $(PC\text{-}LP)$ can be shown to be equivalent to the following, perhaps more natural, linear programming relaxation of the prize-collecting Steiner tree problem, which was used by the algorithm of Bienstock et al. [4]:

$$
\begin{array}{lll}
\text{Min} & \displaystyle\sum_{e\in E} c_e x_e + \sum_{i\neq r}(1-s_i)\pi_i & \\
\text{subject to:} & & \\
& x(\delta(S)) \geq s_i & i\in S; r\notin S \\
& x_e \geq 0 & e\in E \\
& s_i \geq 0 & i\in V; i\neq r.
\end{array}
$$

The dual of $(PC\text{-}LP)$ can be formulated as follows:

$$
(PC\text{-}D) \qquad
\begin{array}{lll}
\text{Max} & \displaystyle\sum_{S: r\notin S} y_S & \\
\text{subject to:} & & \\
& \displaystyle\sum_{S: e\in\delta(S)} y_S \leq c_e & e\in E \\
& \displaystyle\sum_{S\subseteq T} y_S \leq \sum_{i\in T}\pi_i & T\subset V; r\notin T \\
& y_S \geq 0 & S\subset V; r\notin S.
\end{array}
$$

The algorithm for the prize-collecting Steiner tree problem is shown in Figure 9. The basic structure of this algorithm is similar to that of the main algorithm. The algorithm maintains a forest F of edges, which is initially empty. Hence each vertex v is initially in its own connected component. All components except the root r are considered active, and each vertex is initially unmarked. The algorithm loops, in each iteration doing one of two things. First, the algorithm may add an edge between two connected components of F. If the resulting component contains the root r, it becomes inactive; otherwise it is active. Second, the algorithm may decide to "deactivate" a component. Intuitively, a component is deactivated if the algorithm decides it is willing to pay the penalties for all vertices in the component. In this case, the algorithm labels each vertex in the component with the name of the component. The main loop terminates when all connected components of F are inactive. Since in each iteration the sum of the number of components and the number of active components decreases, the loop terminates after at most $2n-1$ iterations. The final step of the algorithm removes as many edges from F as possible while maintaining two properties. First, all unmarked vertices must be connected to the root, since these vertices were never in any deactivated component and the algorithm was never willing to pay the penalty for these vertices. Second, if a vertex with label C is connected to the root, then so is every vertex with label $C' \supseteq C$.

As with the main algorithm, the choices of the algorithm are motivated by the greedy construction of an implicit solution to the dual $(PC\text{-}D)$. Initially all dual variables are set to zero. In each iteration of the main loop, the algorithm increases y_C for all active C by a value ϵ which is as large as possible without violating the two types of packing constraints of $(PC\text{-}D)$: $\sum_{S:e\in\delta(S)} y_S \leq c_e$ for all $e \in E$, and $\sum_{S\subseteq T} y_S \leq \sum_{i\in T} \pi_i$ for all $T \subset V$. Increasing the y_C for active C by ϵ will cause one of the packing constraints to become tight. If one of the first kind of constraints becomes tight, then it becomes tight for some edge e between two connected components of the current forest F; hence we add this edge to F. If one of the second kind of constraints becomes tight, then it becomes tight for some active component C. In this case, the algorithm chooses to deactivate C.

We claim that the algorithm shown in Figure 9 behaves exactly in the manner described above. The claim follows straightforwardly from the algorithm's construction of y and F, and from the fact that $d(i) = \sum_{S:i\in S} y_S$ and $w(C) = \sum_{S\subseteq C} y_S$ at the beginning of each iteration. Note that the algorithm keeps track of the activity of component C by setting $\lambda(C) = 1$ if and only if component C is active.

Let Z^*_{PCLP} and Z^*_{PCIP} be the optimal solutions to $(PC\text{-}LP)$ and $(PC\text{-}IP)$ respectively. Obviously $\sum_{S\subset V} y_S \leq Z^*_{PCLP} \leq Z^*_{PCIP}$. In a manner analogous to that of Theorem 2.4, we will show the following theorem.

THEOREM 4.1. *The algorithm in Figure 9 produces a set of edges F' and a set of vertices X whose incidence vectors are feasible for $(PC\text{-}IP)$, and such that*

$$\sum_{e\in F'} c_e + \sum_{i\in X} \pi_i \leq (2 - \frac{1}{n-1}) \sum_{S\subset V} y_S \leq (2 - \frac{1}{n-1}) Z^*_{PCIP}.$$

Hence the algorithm is a $(2-\frac{1}{n-1})$-approximation algorithm for the prize-collecting Steiner tree problem.

Proof. It is not hard to see that the algorithm produces a feasible solution to $(PC\text{-}IP)$, since F' has no non-trivial component not containing r and the component containing r is a tree.

Input: An undirected graph $G = (V, E)$, edge costs $c_{ij} \geq 0$, vertex penalties $\pi_i \geq 0$, and a root vertex r

Output: A tree F', which includes vertex r, and a set of unspanned vertices X

$F \leftarrow \emptyset$
Comment: Implicitly set $y_S \leftarrow 0$ *for all* $S \subset V$
$\mathcal{C} \leftarrow \{\{v\} : v \in V\}$
For each $v \in V$
  Unmark v
  $d(v) \leftarrow 0$
  $w(\{v\}) \leftarrow 0$
  If $v = r$ then $\lambda(\{v\}) \leftarrow 0$ else $\lambda(\{v\}) \leftarrow 1$
While $\exists C \in \mathcal{C} : \lambda(C) = 1$
  Find edge $e = (i, j)$ with $i \in C_p \in \mathcal{C}$, $j \in C_q \in \mathcal{C}$, $C_p \neq C_q$ that minimizes $\epsilon_1 = \frac{c_e - d(i) - d(j)}{\lambda(C_p) + \lambda(C_q)}$
  Find $\tilde{C} \in \mathcal{C}$ with $\lambda(\tilde{C}) = 1$ that minimizes $\epsilon_2 = \sum_{i \in \tilde{C}} \pi_i - w(\tilde{C})$
  $\epsilon = \min(\epsilon_1, \epsilon_2)$
  $w(C) \leftarrow w(C) + \epsilon \cdot \lambda(C)$ for all $C \in \mathcal{C}$
  Comment: Implicitly set $y_C \leftarrow y_C + \epsilon \cdot \lambda(C)$ *for all* $C \in \mathcal{C}$
  For all $v \in C_r \in \mathcal{C}$
    $d(v) \leftarrow d(v) + \epsilon \cdot \lambda(C_r)$
  If $\epsilon = \epsilon_2$
    $\lambda(\tilde{C}) \leftarrow 0$
    Mark all unlabelled vertices of $\tilde{C}$ with label $\tilde{C}$
  else
    $F \leftarrow F \cup \{e\}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_i \cup C_j\} - \{C_i\} - \{C_j\}$
    $w(C_p \cup C_q) \leftarrow w(C_p) + w(C_q)$
    If $r \in C_p \cup C_q$ then $\lambda(C_p \cup C_q) \leftarrow 0$ else $\lambda(C_p \cup C_q) \leftarrow 1$
F' is derived from F by removing as many edges as possible but so that the following two properties hold: (1) every unlabelled vertex is connected to r; (2) if vertex v with label C is connected to r, then so is every vertex with label $C' \supseteq C$.
X is the set of all vertices not spanned by F'.

FIG. 9. *The algorithm for the Prize-Collecting Steiner Tree Problem.*

By the construction of F', each vertex not spanned by F' (i.e., the vertices in X) lies in some component deactivated at some point during the algorithm. Furthermore, if the vertex was in some deactivated component C, then none of the vertices of C are spanned by F'. Using these observations, plus the manner in which components are formed by the algorithm, we can partition the vertices of X into disjoint deactivated components $C_1, \ldots, C_k$. These sets are the maximal labels of the vertices in X. Since each C_j is a deactivated component, it follows that $\sum_{S \subseteq C_j} y_S = \sum_{i \in C_j} \pi_i$, and thus that the inequality to be proven is implied by $\sum_{e \in F'} c_e + \sum_j \sum_{S \subseteq C_j} y_S \leq (2 - \frac{1}{n-1}) \sum_{S \subset V} y_S$. In addition, since $c_e = \sum_{S: e \in \delta(S)} y_S$ for each $e \in F'$ by construction of the algorithm, all we need to prove is that

$$\sum_{e \in F'} \sum_{S: e \in \delta(S)} y_S + \sum_j \sum_{S \subseteq C_j} y_S \leq (2 - \frac{1}{n-1}) \sum_{S \subset V} y_S,$$

or, rewriting terms as in Theorem 2.4,

$$\sum_S y_S |F' \cap \delta(S)| + \sum_j \sum_{S \subseteq C_j} y_S \leq (2 - \frac{1}{n-1}) \sum_{S \subset V} y_S.$$

As in Theorem 2.4, this theorem can be proven by induction on the main loop. Pick any particular iteration, and let $\mathcal{C}$ be the set of active components of the iteration. Let H be the graph formed by considering active and inactive components as vertices and the edges $e \in \delta(C) \cap F'$ for active C as the edges of H. Discard all isolated inactive vertices. Let N_a denote the set of active vertices in H, N_i the set of inactive vertices, N_d the set of active vertices corresponding to active sets contained in some C_j, and d_v the degree of a vertex v in H. Notice that $N_d = \{v \in N_a : d_v = 0\}$. In this iteration, the increase in the left-hand side of the inequality is $\epsilon(\sum_{v \in N_a} d_v + |N_d|)$ while the increase in the right-hand side of the inequality is $\epsilon(2 - \frac{1}{n-1})|N_a|$. Thus we would like to prove that $(\sum_{v \in N_a} d_v + |N_d|) \leq (2 - \frac{1}{n-1})|N_a|$. Note that the degree of any vertex corresponding to an active set in some C_j is zero. Hence if we can show that $\sum_{v \in N_a - N_d} d_v \leq (2 - \frac{1}{n-1})|N_a - N_d|$, then the proof will be complete.

To do this, we show that all but one of the leaves of H must be active vertices. Suppose that v is an inactive leaf of H, adjacent to edge e, and let C_v be the inactive component corresponding to v. Further suppose that C_v does not contain the root r. Since C_v is inactive and does not contain r, it must have been deactivated. Because C_v is deactivated, no vertex in C_v is unlabelled; furthermore, since v is a leaf, no vertex in C_v can lie on the path between the root and a vertex which must be connected to the root. By the construction of F', then, $e \notin F'$, which is a contradiction. Therefore, there can be at most one inactive leaf, which must correspond to the component containing r.

Then

$$\begin{aligned}
\sum_{v \in N_a - N_d} d_v &\leq \sum_{v \in (N_a - N_d) \cup N_i} d_v - \sum_{v \in N_i} d_v \\
&\leq 2(|(N_a - N_d) \cup N_i| - 1) - (2|N_i| - 1) \\
&= 2|N_a - N_d| - 1 \\
&\leq (2 - \frac{1}{n-1})|N_a - N_d|.
\end{aligned}$$

The inequality holds since all but one of the spanned inactive vertices has degree at least two, and since the number of active components is always at most $n - 1$. □

The algorithm can be implemented in $O(n^2 \log n)$ time, by using the same techniques as were given for the main algorithm. In this case, we must also keep track of the time at which we expect each component to deactivate, and put this time into the priority queue. The only other difference from the main algorithm is the final step in which edges are deleted. This step can be implemented in $O(n^2)$ time: first we perform a depth-first search from every unmarked vertex to the root, and "lock" all the edges and vertices on this path. We then look at all the deactivated components corresponding to the labels of "locked" vertices. If one of these contains an unlocked vertex, we perform a depth-first search from the vertex to the root and lock all the edges and vertices on the path. We continue this process until each locked vertex is in deactivated components that only contain locked vertices. We then eliminate all unlocked edges. This procedure requires at most n $O(n)$ time depth-first searches.

Input: An undirected graph $G = (V, E)$, edge costs $c_{ij} \geq 0$, vertex penalties $\pi_i \geq 0$, and a root vertex r

Output: A tour T', which includes vertex r, and a set of unspanned vertices X

1 Apply the prize-collecting Steiner tree algorithm to the problem instance with graph G, edge costs c, root r, and penalties $\pi'_i = \pi_i/2$.

2 Duplicate the edges F' of the Steiner tree returned to form an Eulerian graph T.

3 Shortcut T to form a tour T'. Let X be all vertices not in the tour.

FIG. 10. *The algorithm for the Prize-Collecting Traveling Salesman Problem.*

4.3.2. The Prize-Collecting Traveling Salesman Problem. In order to solve the prize-collecting TSP given that edge costs obey the triangle inequality, we use the algorithm shown in Figure 10. Note that the algorithm uses the above algorithm for the prize-collecting Steiner tree problem with penalties $\pi'_i = \pi_i/2$. To see that the algorithm is a $(2 - \frac{1}{n-1})$-approximation algorithm, we need to consider the following linear programming relaxation of the problem:

$$
\begin{array}{lll}
\text{Min} & \displaystyle\sum_{e \in E} c_e x_e + \sum_{T \subset V; r \notin T} z_T (\sum_{i \in T} \pi_i) & \\
\text{subject to:} & & \\
& \displaystyle x(\delta(S)) + 2 \sum_{T \supseteq S} z_T \geq 2 & r \notin S \\
& x_e \geq 0 & e \in E \\
& z_T \geq 0 & T \subset V; r \notin T.
\end{array}
$$

This linear program is a relaxation of an integer program similar to $(PC\text{-}IP)$ in which $z_T = 1$ for the set of vertices T not visited by the tour, and $z_T = 0$ otherwise. We relax the constraint that each vertex in the tour be visited twice to the constraint that each vertex be visited at least twice. The dual of the linear programming relaxation is

$$
\begin{array}{lll}
\text{Max} & \displaystyle 2 \sum_{S: r \notin S} y_S & \\
\text{subject to:} & & \\
& \displaystyle \sum_{S: e \in \delta(S)} y_S \leq c_e & e \in E \\
& \displaystyle 2 \sum_{S \subseteq T} y_S \leq \sum_{i \in T} \pi_i & T \subset V, r \notin T \\
& y_S \geq 0 & S \subset V, r \notin S.
\end{array}
$$

Notice that this dual is very similar to $(PC\text{-}D)$. The dual solution generated by the algorithm for the prize-collecting Steiner tree for penalties π' will be feasible for the dual program above with penalties π. By duality, $2 \sum_{S \subset V} y_S \leq Z^*_{PCTSP}$, where Z^*_{PCTSP} is the cost of the optimal solution to the prize-collecting TSP. Given a solution F' and X to the prize-collecting Steiner tree problem, the cost of our solution to the prize-collecting TSP is at most $2 \sum_{e \in F'} c_e + \sum_{i \in X} \pi_i = 2(\sum_{e \in F'} c_e + \sum_{i \in X} \pi'_i)$.

Theorem 4.1 shows that $\sum_{e\in F'} c_e + \sum_{i\in X} \pi'_i \leq (2-\frac{1}{n-1})\sum_{S\subset V} y_S$, so that

$$2(\sum_{e\in F'} c_e + \sum_{i\in X} \pi'_i) \leq 2(2-\frac{1}{n-1})\sum_{S\subset V} y_S \leq (2-\frac{1}{n-1})Z^*_{PCTSP}.$$

Thus the cost of the solution found by the algorithm is within $(2-\frac{1}{n-1})$ of optimal.

5. Concluding Remarks. The approximation techniques described in the previous sections have been applied to a number of related problems since the appearance of a preliminary version of this paper [16]. Saran, Vazirani, and Young [31] showed how to use our techniques to derive an approximation algorithm for the minimum-cost 2-edge-connected graph problem. Their algorithm has a performance guarantee of 3, equal to the performance guarantee of an earlier algorithm of Frederickson and Ja'Ja' [10] for the same problem. Klein and Ravi [22] demonstrated a 3-approximation algorithm for solving (IP) for proper functions $f : 2^V \rightarrow \{0,2\}$. Building on some ideas of Ravi and Klein, Williamson, Goemans, Mihail, and Vazirani [38] devised an approximation algorithm to solve (IP) for general proper functions $f : 2^V \rightarrow \mathbb{N}$ in which the disjointness condition is replaced by the more general condition $f(A\cup B) \leq \max(f(A), f(B))$ for disjoint A, B. The performance guarantee of the algorithm is at most $2k$, where $k = \max_S f(S)$, but can be lower depending on the values taken on by f. The algorithm depends on showing that the techniques of this paper can be extended to approximate uncrossable functions, as defined in Section 4. Goemans, Goldberg, Plotkin, Shmoys, Tardos, and Williamson [14] have shown how to improve the performance guarantee of this algorithm to $2(1+\frac{1}{2}+\cdots+\frac{1}{k})$. Gabow, Goemans, and Williamson [12] have shown how to efficiently implement the algorithm of Williamson et al. A consequence of the implementation of Gabow et al. is a $O(n^2 + n\sqrt{m \log\log n})$ implementation for our main algorithm. Finally, we have implemented the 2-approximation algorithm for Euclidean matching problems [37]. The performance of the algorithm in this case seems to be much better than the theoretical bounds given here: on 1,400 random and structured instances of up to 131,072 vertices, the algorithm was never more than 4% away from optimal.

Acknowledgements. The authors would like to thank David Shmoys for extensive comments on a draft of this paper.

REFERENCES

[1] A. Agrawal, P. Klein, and R. Ravi, *When trees collide: An approximation algorithm for the generalized Steiner problem on networks*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, 1991, pp. 134–144. To appear in *SIAM J. Comput.*

[2] E. Balas, *The prize collecting traveling salesman problem*, Networks, 19 (1989), pp. 621–636.

[3] P. Berman and V. Ramaiyer, *Improved approximations for the Steiner tree problem*, in Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 325–334.

[4] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson, *A note on the prize collecting traveling salesman problem*, Mathematical Programming, 59 (1993), pp. 413–420.

[5] V. Chvatal, *A greedy heuristic for the set-covering problem*, Mathematics of Operations Research, 4 (1979), pp. 233–235.

[6] G. Cornuéjols and W. Pulleyblank, *A matching problem with side constraints*, Discrete Mathematics, 29 (1980), pp. 135–159. Papadimitriou's result appears in this paper.

[7] J. Edmonds, *Maximum matching and a polyhedron with 0,1-vertices*, Journal of Research of the National Bureau of Standards B, 69B (1965), pp. 125–130.

[8] J. Edmonds and E. Johnson, *Matching: A well-solved class of integer linear programs*, in Proceedings of the Calgary International Conference on Combinatorial Structures and Their

Applications, R. Guy, H. Hanani, N. Sauer, and J. Schonheim, eds., Gordon and Breach, 1970, pp. 82–92.

[9] J. Edmonds and E. L. Johnson, *Matching, Euler tours and the Chinese postman*, Mathematical Programming, 5 (1973), pp. 88–124.

[10] G. N. Frederickson and J. Ja'Ja', *Approximation algorithms for several graph augmentation problems*, SIAM Journal on Computing, 10 (1981), pp. 270–283.

[11] H. N. Gabow, *Data structures for weighted matching and nearest common ancestors with linking*, in Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, 1990, pp. 434–443.

[12] H. N. Gabow, M. X. Goemans, and D. P. Williamson, *An efficient approximation algorithm for the survivable network design problem*, in Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization, 1993, pp. 57–74.

[13] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for general graph-matching problems*, Journal of the ACM, 38 (1991), pp. 815–853.

[14] M. Goemans, A. Goldberg, S. Plotkin, D. Shmoys, E. Tardos, and D. Williamson, *Improved approximation algorithms for network design problems*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 223–232.

[15] M. X. Goemans and D. J. Bertsimas, *Survivable networks, linear programming relaxations and the parsimonious property*, Mathematical Programming, 60 (1993), pp. 145–166.

[16] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, in Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 307–316. To appear in *SIAM J. Comput.*

[17] M. X. Goemans and D. P. Williamson, *Approximating minimum-cost graph problems with spanning tree edges*, Operations Research Letters, 16 (1994), pp. 183–189.

[18] C. Imielińska, B. Kalantari, and L. Khachiyan, *A greedy heuristic for a minimum-weight forest problem*, Operations Research Letters, 14 (1993), pp. 65–71.

[19] D. S. Johnson, *Approximation algorithms for combinatorial problems*, Journal of Computer and System Sciences, 9 (1974), pp. 256–278.

[20] M. Jünger and W. Pulleyblank, *New primal and dual matching heuristics*, Research Report 91.105, Universität zu Köln, 1991.

[21] M. Jünger and W. Pulleyblank, *Geometric duality and combinatorial optimization*, in Jahrbuch Überblicke Mathematik 1993, S. Chatterji, B. Fuchssteiner, U. Kulisch, and R. Liedl, eds., Vieweg, Braunschweig, Germany, 1993, pp. 1–24. Also appears as IBM Research Report RC 17863 (#78620).

[22] P. Klein and R. Ravi, *When cycles collapse: A general approximation technique for constrained two-connectivity problems*, in Proceedings of the Third MPS Conference on Integer Programming and Combinatorial Optimization, 1993, pp. 39–55. Also appears as Brown University Technical Report CS-92-30.

[23] J. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings of the American Mathematical Society, 7 (1956), pp. 48–50.

[24] G. Laporte, *Location-routing problems*, in Vehicle routing: Methods and studies, B. L. Golden and A. A. Assad, eds., North-Holland, Amsterdam, 1988, pp. 163–197.

[25] G. Laporte, Y. Nobert, and P. Pelletier, *Hamiltonian location problems*, European Journal of Operations Research, 12 (1983), pp. 82–89.

[26] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, *The point-to-point delivery and connection problems: Complexity and algorithms*, Discrete Applied Mathematics, 36 (1992), pp. 267–292.

[27] L. Lovász, *On the ratio of optimal integral and fractional covers*, Discrete Mathematics, 13 (1975), pp. 383–390.

[28] T. Nicholson, *Finding the shortest route between two points in a network*, Computer Journal, 9 (1966), pp. 275–280.

[29] D. A. Plaisted, *Heuristic matching for graphs satisfying the triangle inequality*, Journal of Algorithms, 5 (1984), pp. 163–179.

[30] E. M. Reingold and R. E. Tarjan, *On a greedy heuristic for complete matching*, SIAM Journal on Computing, 10 (1981), pp. 676–681.

[31] H. Saran, V. Vazirani, and N. Young, *A primal-dual approach to approximation algorithms for network Steiner problems*, in Proceedings of Indo-US workshop on Cooperative Research in Computer Science, 1992, pp. 166–168.

[32] K. J. Supowit, D. A. Plaisted, and E. M. Reingold, *Heuristics for weighted perfect matching*, in Proceedings of the 12th Annual ACM Symposium on Theory of Computing, 1980, pp. 398–419.

[33] R. E. Tarjan, *Efficiency of a good but not linear set union algorithm*, Journal of the ACM,

22 (1975), pp. 215–225.

[34] P. Vaidya. Personal communication, 1991.

[35] O. Vornberger, *Complexity of path problems in graphs*, PhD thesis, Universität-GH-Paderborn, 1979.

[36] D. P. Williamson, *On the design of approximation algorithms for a class of graph problems*, PhD thesis, MIT, Cambridge, MA, September 1993. Also appears as Tech Report MIT/LCS/TR-584.

[37] D. P. Williamson and M. X. Goemans, *Computational experience with an approximation algorithm on large-scale Euclidean matching instances*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 355–364.

[38] D. P. Williamson, M. X. Goemans, M. Mihail, and V. V. Vazirani, *A primal-dual approximation algorithm for generalized Steiner network problems*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, 1993, pp. 708–717. To appear in *Combinatorica*.

[39] P. Winter, *Steiner problem in networks: a survey*, Networks, 17 (1987), pp. 129–167.

[40] A. Zelikovsky, *An 11/6-approximation algorithm for the network Steiner problem*, Algorithmica, 9 (1993), pp. 463–470.