

# The Prize Collecting Steiner Tree Problem: Theory and Practice

David S. Johnson \*

Maria Minkoff †

Steven Phillips ‡

## Abstract

We consider variants on the Prize Collecting Steiner Tree problem and on the primal-dual 2-approximation algorithm devised for it by Goemans and Williamson. We introduce an improved pruning rule for the algorithm that is slightly faster and provides solutions that are at least as good and typically significantly better. On a selection of real-world instances whose underlying graphs are county street maps, the improvement in the standard objective function ranges from 1.7% to 9.2%. Substantially better improvements are obtained for the complementary “net worth” objective function and for randomly generated instances. We also show that modifying the growth phase of the Goemans-Williamson algorithm to make it independent of the choice of root vertex does not significantly affect the algorithm’s worst-case guarantee or behavior in practice. The resulting algorithm can be further modified so that, without an increase in running time, it becomes a 2-approximation algorithm for finding the best subtree over *all* choices of root.

In the second part of the paper, we consider *quota* and *budget* versions of the problem. In the first, one is looking for the tree with minimum edge cost that contains vertices whose total prize is at least a given quota; in the second one is looking for the tree with maximum prize, given that the total edge cost is within a given budget. The quota problem is a generalization of the  $k$ -MST problem, and we observe how constant-factor approximation algorithms for that problem can be extended to it. We also show how a  $(5 + \epsilon)$ -approximation algorithm for the (unrooted) budget problem can be derived from Garg’s 3-approximation algorithm for the  $k$ -MST. None of these algorithms are likely to be used in practice, but we show how the general approach behind them (which involves performing multiple runs of the Goemans-Williamson algorithm using an increasing sequence of prize-multipliers) can be incorporated into a practical heuristic. We also uncover some surprising properties of the cost/prize tradeoff curves generated (and used) by this approach.

## 1 Problem Definitions

In the *Prize Collecting Steiner Tree* (PCST) problem, one is given a graph  $G = (V, E)$ , a non-negative *edge cost*  $c(e)$  for each edge  $e \in E$ , a non-negative *vertex prize*  $p(v)$  for each vertex  $v \in V$ , and a specified *root vertex*  $v_0 \in V$ . In this paper we shall consider four different optimization problems based on this scenario, the first being the one initially studied in [6, 7]:

1. The *Goemans-Williamson Minimization* problem: Find a subtree  $T' = (V', E')$  of  $G$  that minimizes the cost of the edges in the tree plus the prizes of the vertices *not* in the tree, i.e., that minimizes

$$GW(T') = \sum_{e \in E'} c(e) + \sum_{v \notin V'} p(v)$$

2. The *Net Worth Maximization* problem: Find a subtree  $T'$  that maximizes

$$NW(T') = \sum_{v \in V'} p(v) - \sum_{e \in E'} c(e)$$

3. The *Quota* problem: Given a *prize quota*  $Q > 0$ , find a subtree  $T'$  that minimizes  $\sum_{e \in E'} c(e)$ , subject to  $\sum_{v \in V'} p(v) \geq Q$ .
4. The *Budget* problem: Given an *edge budget*  $B > 0$ , find a subtree  $T'$  that maximizes  $\sum_{v \in V'} p(v)$ , subject to  $\sum_{e \in E'} c(e) \leq B$ .

In addition, for each problem we have both the *rooted* variant, in which  $v_0$  must be contained in  $T'$ , and the *unrooted* variant, in which  $T'$  can be *any* subtree.

In [6, 7], Goemans and Williamson present an  $O(n^2 \log n)$ -time primal-dual approximation algorithm for the rooted version of GW-Minimization that is guaranteed to be within a factor of  $2 - \frac{1}{n-1}$  of optimal, where  $n = |V|$ . (In what follows, we shall refer to this algorithm as the *GW-algorithm* for short.) The perhaps more natural NW-Maximization variant is equivalent to GW-Minimization as far as optimization is concerned, since for all subtrees  $T'$ ,  $NW(T') + GW(T')$  must equal the total prize (over all vertices of  $V$ ). The two problems are not equivalent with respect to approximation, however: It is NP-hard to approximate the optimum Net Worth to within any constant factor [4].

As to the Quota problem, note that if it is restricted to instances in which all vertices have prize 1, we obtain the well-studied  $k$ -MST problem, for which the current best approximation results are a factor-of-3 algorithm for the rooted case [5] and a factor-of-2.5 algorithm for the unrooted case [2]. Interestingly,

\*AT&T Labs, Room C239, 180 Park Avenue, Florham Park, NJ 07932. Email: dsj@research.att.com

†MIT Lab. for Computer Science, 545 Tech Square, Cambridge, MA 02139. Email: mariam@theory.lcs.mit.edu

‡AT&T Labs, Room A003, 180 Park Avenue, Florham Park, NJ 07932. Email: phillips@research.att.com

these algorithms depend in a crucial way on the GW-algorithm mentioned above, as we shall explain in more detail below. We know of no previous approximation results for the budget problem, rooted or unrooted.

## 2 Motivating Applications: Local Access Network Design

The general PCST problem has a variety of applications, but the one that motivates this paper is its application to local access network design. For instance, suppose we wish to build a fiber-optic network for providing broadband Internet connections to businesses and apartment complexes. Here the graph might correspond to a street map, since local-access fiber is typically laid along streets. The vertices of the graph would be the street intersections and the locations of potential customer buildings, which as a first approximation we can assume lie on the streets. The “prize” associated with the vertex representing a building is some estimate of the revenue to be obtained by connecting up that building. (Vertices corresponding to intersections rather than buildings have 0 prize.) The edges are then the street segments between vertices, and the cost of an edge is the cost of laying a fiber-optic cable along that segment. This is typically dominated by labor and right-of-way charges, not by the cost of the cable itself, so we can assume that cable capacity is not an issue.

A similar application arises in designing optical fiber backhaul networks for fixed wireless systems. Here the prize locations would simply be the telephone polls where antennas might be placed, with the prize for a location being a measure of the number of potential customers reachable from that antenna site. Edge costs would be as before.

Of the four main PCST variants, although GW-minimization is the most studied, it makes the least economic sense in terms of these applications. Assuming that costs and prizes can be expressed in comparable terms, NW-maximization is more plausible, since it corresponds to finding the most profitable network. However, spending \$1,000,000 to make a net profit of \$10 is probably not as sensible as spending \$10 to make a net profit of \$5, so this variant too has its drawbacks. Moreover, in our applications the costs and prizes are not so easily compared, since the first correspond to actual expenditures and the second only to potential revenue (and in the fixed wireless case, only to an indirect measure of that). Thus the Budget and Quota variants are the ones most likely to be of practical interest, and in particular the related tradeoff curves: As a function of prize quota, what is the least edge-cost we need incur and, as a function of edge cost budget, what is the most prize we can collect?

## 3 Modifying the GW-Algorithm

We begin by studying the GW-algorithm in the context of GW-minimization and NW-maximization. Although we have argued the irrelevance of these problems to our application, the GW-algorithm is a key subroutine in algorithms for the more-relevant Quota and Budget problems, so it is useful to know how it performs (and how its performance can be improved). Moreover, there has been no serious previous study of the GW-algorithm’s empirical behavior in its original context, and so our experimental results can be viewed as a first step toward putting the corresponding theoretical results in perspective.

### 3.1 The Two Phases of the GW-Algorithm.

The GW-algorithm for the PCST works as follows: In the initial *growth phase*, we maintain a set of *components* (initially each vertex is its own component), each component  $C$  having a *surplus*  $s(C) \geq 0$  (initially a vertex’s prize). Component  $C$  is *active* if  $s(C) > 0$  and the root vertex  $v_0$  is not in  $C$ . In addition, each *active* edge  $e$  (one not joining two vertices in the same component) has a *deficit*  $d(e) > 0$ , initially equal to its cost.

As time passes, we view the surpluses as paying down the deficits as follows: in each positive increment  $\Delta$  of time, the surpluses of all active components are reduced by  $\Delta$ , the deficit of any active edge adjacent to a single active component is reduced by  $\Delta$ , and the deficit of any active edge adjacent to two active components is reduced by  $2\Delta$ . This continues until one of the following two events occurs: (1) an active edge’s deficit is reduced to 0, or (2) an active component’s surplus is reduced to 0. We process all events of type (1) first (ties broken arbitrarily), merging the two components linked by the edge and giving the new merged component the sum of the surpluses for the two components being merged. We then process the events of type (2), deactivating all the newly surplus-free components.

This process continues until either all vertices are in the root component or all non-root component surpluses are 0. We then begin the *pruning phase*, which we shall call “GW-pruning.” In this phase vertices are removed from the root component based on the order in which certain events occurred during the growth phase. This is a bit complicated and requires special labeling and bookkeeping during the growth phase, but for this paper we do not need to know the precise details.

**3.2 Strong Pruning.** Our first observation is that there is a much more straightforward way to prune than GW-pruning, one that provably dominates it, and hence can be substituted for it without any loss in the worst-case guarantees for the GW-algorithm. Suppose we are

given a subtree  $T'$  of  $G$  with root  $v_0$ , and let us view all the edges of  $T'$  as directed away from the root. Initially define the net worth  $nw(v)$  for a vertex of  $T'$  to simply be its prize  $p(v)$ . *Strong pruning* can be defined by the following recursive program, initially applied to the root of the tree.

```

StrongPrune( $v$ )
  For all children  $u$  of  $v$ , do the
  following:
    StrongPrune( $u$ ).
    If  $c(\{v, u\}) \geq nw(u)$ , remove the
      edge  $\{v, u\}$  and the subtree
      rooted at  $u$  from  $T'$ .
    Otherwise, set
       $nw(v) = nw(v) + nw(u) - c(\{v, u\})$ .
  Return.

```

It is easy to see that strong pruning can be performed in  $O(n)$  time. Moreover, since it does not require any knowledge of what went on during the growth phase, none of the detailed labeling and record-keeping required for GW-pruning need be performed during that phase, which simplifies the programming task. As to the quality of the tree constructed, the following Theorem is easy to prove by induction.

**THEOREM 3.1.** *Let  $T$  be any subtree of  $G$  rooted at  $v_0$ , let  $T'$  be any subtree of  $T$  that contains  $v_0$ , and let  $T_{SP}$  be the tree obtained from  $T$  by strong pruning. Then  $GW(T_{SP}) \leq GW(T')$ .*

Note also that, assuming as we do that prizes are non-negative,  $NW(T_{SP})$  must be nonnegative. This need not hold for the tree generated by GW-pruning.

**3.3 Experimental Background: Machines and Instances.** All our experiments were run on single 194 or 196 Mhz MIPS R10000 processors contained in shared-memory multiprocessor SGI Power Challenge computers with 2 or 6 gigabytes of random access memory, more than enough for our experiments to fit in main memory. Data and instruction cache sizes were 32 kilobytes. This is sufficiently small that our running times were likely affected by (not-easily-quantifiable) caching effects. (Fortunately, highly detailed conclusions about running time are not needed here.) The operating system was IRIX 6.2, and the programs were written in C and compiled using the SGI-supplied compiler.

We report on three classes of instances. The first consists of instances whose graphs were derived from the street maps of 12 U.S. counties of varying demographic characteristics, ranging from Silicon Valley to Chicago to suburbs of Washington, D.C., and from 14,000 to 76,000 vertices. Edge costs are approximately the

Euclidean distance in miles, and prize vertices (ones with nonzero prize) and their associated prizes are placed so as to reflect demographics. Basic statistics for these instances, including county name, numbers of vertices and edges, percent of vertices with nonzero prize, and the ratio of the average nonzero prize to the average edge cost, are reported in Table 1.

The second class of instances are random geometric instances designed to have a local structure somewhat similar to that of our street map instances. Vertices correspond to random points in the unit square. There is an edge between two vertices if their distance is no more than  $1.6/\sqrt{n}$ , and the cost of an edge is the Euclidean distance between the two points. A vertex received a nonzero prize with probability 0.15, and the prize values were chosen uniformly between 0 and  $3/\sqrt{n}$ . These dependencies on  $n$  were chosen so that the resulting graphs would have roughly constant expected average degree and (average non-zero prize)/(average edge cost) ratios, as can be seen from the statistics presented in Table 1. Values of  $n$  were chosen going from 100 to 25,600 by factors of 4, and several examples were produced for each value of  $n$ .

Our final class was also designed to have constant expected degree and prize/cost ratio, but beyond that was unstructured. Two vertices have an edge between them with probability  $6/n$ , a vertex has nonzero prize with probability  $1/4$ , and both edge lengths and vertex prizes are chosen uniformly between 0 and 1. Details are once more included in Table 1.

The two random classes were chosen to share some of the properties of our proposed application (bounded vertex degree, and prize/cost ratios) while potentially providing insight into the dependence of results on  $n$  in different parts of the space of possible instances. Other distributions might also provide insight, although we note that it takes some thought to design classes for which the GW-algorithm does not degenerate to claiming either all the prize or only the prize at the root (which comes for free).

### 3.4 Experimental Results: Strong Pruning.

Our results are summarized in Tables 1 and 2, with running times given in Table 1 and solution quality data given in Table 2. Note that the difference in running time between GW-pruning and strong pruning is insignificant, even though the former is more complicated and adds a bit of extra overhead to the growth phase. This is because pruning (and the extra growth phase overhead) takes time  $O(n)$ , whereas the overall growth phase has worst-case time  $O(n^2 \log n)$ . For both classes of random graphs, the measured running times appear to be trending towards this worst-case bound,

| Streetmap Graphs |    | Vertices | Edges   | % Prizes | Ave P/C | Ave. % Vertices in Tree | Rooted Growth |                | Unrooted Growth |              |
|------------------|----|----------|---------|----------|---------|-------------------------|---------------|----------------|-----------------|--------------|
| County           |    |          |         |          |         |                         | GW Pruning    | Strong Pruning | Strong Pruning  | Best Pruning |
| Cook             | IL | 76,115   | 128,375 | 8.6      | 7.6     | 24.6                    | 335 sec       | 334 sec        | 601 sec         | 599 sec      |
| Dallas           | TX | 47,082   | 75,593  | 9.0      | 4.0     | 18.4                    | 108 sec       | 110 sec        | 157 sec         | 158 sec      |
| Wayne            | MI | 41,433   | 69,917  | 9.7      | 3.0     | 19.3                    | 90 sec        | 89 sec         | 134 sec         | 134 sec      |
| King             | WA | 43,902   | 67,344  | 14.1     | 1.1     | 10.1                    | 33 sec        | 32 sec         | 46 sec          | 48 sec       |
| Suffolk          | NY | 43,092   | 66,645  | 10.5     | 2.4     | 15.0                    | 65 sec        | 65 sec         | 95 sec          | 95 sec       |
| Nassau           | NY | 32,067   | 51,676  | 5.7      | 10.3    | 25.8                    | 53 sec        | 52 sec         | 77 sec          | 77 sec       |
| Oakland          | MI | 28,305   | 43,223  | 15.4     | 1.8     | 15.9                    | 30 sec        | 28 sec         | 45 sec          | 45 sec       |
| Franklin         | OH | 21,765   | 34,216  | 12.8     | 2.8     | 20.1                    | 16 sec        | 15 sec         | 21 sec          | 22 sec       |
| Duval            | FL | 20,995   | 32,890  | 13.6     | 2.3     | 16.4                    | 12 sec        | 13 sec         | 20 sec          | 21 sec       |
| Jefferson        | KY | 15,191   | 23,853  | 12.9     | 1.9     | 12.4                    | 8 sec         | 8 sec          | 13 sec          | 13 sec       |
| Montgomery       | MD | 14,918   | 22,518  | 15.5     | 4.3     | 27.8                    | 7 sec         | 7 sec          | 11 sec          | 11 sec       |
| San Mateo        | CA | 14,429   | 22,442  | 13.7     | 4.2     | 21.5                    | 8 sec         | 7 sec          | 10 sec          | 10 sec       |

| Random Graphs |          |         | Ave % Prizes | Ave Degree | Ave P/C | Ave % Vertices in Tree | Rooted Growth |                | Unrooted Growth |              |
|---------------|----------|---------|--------------|------------|---------|------------------------|---------------|----------------|-----------------|--------------|
| Type          | Vertices | # Inst. |              |            |         |                        | GW Pruning    | Strong Pruning | Strong Pruning  | Best Pruning |
| Geometric     | 100      | 11      | 12.6         | 6.9        | 1.32    | 3.7                    | .01 sec       | .01 sec        | .02 sec         | .02 sec      |
|               | 400      | 11      | 14.9         | 7.5        | 1.49    | 5.5                    | .09 sec       | .09 sec        | .10 sec         | .10 sec      |
|               | 1,600    | 11      | 14.7         | 7.8        | 1.39    | 7.5                    | .52 sec       | .51 sec        | .59 sec         | .60 sec      |
|               | 6,400    | 5       | 15.2         | 7.9        | 1.43    | 10.5                   | 4.1 sec       | 4.0 sec        | 6.1 sec         | 6.2 sec      |
|               | 25,600   | 5       | 15.2         | 8.0        | 1.41    | 10.5                   | 87.8 sec      | 85.7 sec       | 173.7 sec       | 176.9 sec    |
| Un-Structured | 100      | 5       | 29.4         | 6.3        | .99     | 35.3                   | .02 sec       | .02 sec        | .02 sec         | .02 sec      |
|               | 400      | 5       | 26.4         | 5.9        | .98     | 31.8                   | .13 sec       | .13 sec        | .17 sec         | .17 sec      |
|               | 1,600    | 5       | 24.9         | 6.0        | .99     | 31.2                   | 1.4 sec       | 1.4 sec        | 2.0 sec         | 2.0 sec      |
|               | 6,400    | 5       | 25.2         | 6.0        | 1.02    | 31.8                   | 33.3 sec      | 33.9 sec       | 51.7 sec        | 52.2 sec     |
|               | 25,600   | 1       | 25.4         | 6.0        | 1.00    | 31.3                   | 1344 sec      | 1356 sec       | 1923 sec        | 1780 sec     |

Table 1: Statistics and running times for out testbed of instances and variants on the GW algorithm. The “% Prizes” column gives the percentage of vertices with nonzero prizes and the “Ave P/C” column gives the ratio of the average nonzero prize to the average edge cost. The “Ave. % Vertices in Tree” column gives the average percentage of vertices included in the tree generated using rooted strong pruning.

although it is difficult to disentangle cache-miss effects from algorithmic ones on the larger instances, and even our largest instances take less than a half-hour on our relatively slow (by today’s standards) machine. A similar running time growth appears also to occur for the street map instances, although here the picture is less clear. Given the wide variation in geography and population density among the counties covered, counties with roughly the same numbers of vertices and edges can yield widely different running times. For instance, King, Suffolk, and Wayne Counties all have graphs with around 43,000 vertices and 67,000 edges, but the running times for the last two are roughly 2 and 3 times as long as that for the first. This is presumably because of the differing percentage of prize vertices among the three graphs and the different ratios of average prize to average edge cost, which lead to significantly different sizes for the output trees.

We suspect that, although it was not needed for the experiments reported here, a significant improvement in running time could be obtained by more careful engineering of the data structures used for the growth phase. The major work of the growth phase involves updating the status of edges as to (1) which components

contain their endpoints, (2) whether another edge that joins the same components is effectively shorter, and (3) the status (active or inactive) of their endpoint components. During a component merge, our implementation currently takes time proportional to the number of undominated edges leaving the two components involved. With a more intricate implementation, this should be reducible to something proportional to the out-degree of the smaller component, which might make a substantial difference on instances such as ours, where many components are initially inactive ones consisting of a single prize-less vertex with low degree.

As to tree quality, the average improvement in the GW objective function due to strong pruning is significant: GW pruning yields solutions that are 1.7 to 9.2% worse for the street map instances. Comparable improvements occur for the random unstructured instances, and even bigger ones (10% and more) occur for the random geometric graphs. Still bigger improvements occur for the NW objective function: over 20% for three of the street map graphs and over 200% for all the random geometric graphs. (For these the GW-algorithm produces solutions with negative Net Worth.)

Note also, that even if one takes the best solution

| Streetmap Graphs |    | Rooted Growth  |        |            |        |        | Unrooted Growth |        |
|------------------|----|----------------|--------|------------|--------|--------|-----------------|--------|
|                  |    | Strong Pruning |        | GW Pruning |        |        | Strong Pruning  |        |
|                  |    | +MST           | "Best" | Average    | "Best" | Ave NW | Average         | Best   |
| Cook             | IL | -.35           | -.07%  | +3.0%      | +2.8%  | +1.7%  | +.04%           | +.03%  |
| Dallas           | TX | -.21           | -.21%  | +4.9%      | +4.5%  | +6.2%  | -.01%           | -.05%  |
| Wayne            | MI | -.18           | -.12%  | +5.2%      | +5.0%  | +14.5% | -.02%           | -.07%  |
| King             | WA | -.30           | -2.76% | +1.7%      | -1.1%  | +1.6%  | -.02%           | -2.75% |
| Suffolk          | NY | -.23           | -1.05% | +6.4%      | +5.6%  | +21.9% | -.01%           | -1.03% |
| Nassau           | NY | -.32           | -.06%  | +1.9%      | +1.7%  | +1.3%  | -.03%           | -.05%  |
| Oakland          | MI | -.34           | -.31%  | +7.5%      | +6.3%  | +28.3% | -.01%           | -.23%  |
| Franklin         | OH | -.24           | -.23%  | +5.8%      | +5.4%  | +11.9% | -.04%           | -.20%  |
| Duval            | FL | -.35           | -.82%  | +5.3%      | +4.7%  | +14.3% | -.01%           | -.61%  |
| Jefferson        | KY | -.18           | -.36%  | +9.2%      | +8.6%  | +46.9% | -.00%           | -.27%  |
| Montgomery       | MD | -.34           | -.21%  | +2.3%      | +1.5%  | +1.8%  | -.07%           | -.19%  |
| San Mateo        | CA | -.16           | -2.24% | +2.6%      | +.4%   | +1.5%  | -.09%           | -2.24% |

| Random Graphs |        | Rooted Growth  |         |            |        |         | Unrooted Growth |         |
|---------------|--------|----------------|---------|------------|--------|---------|-----------------|---------|
|               |        | Strong Pruning |         | GW Pruning |        |         | Strong Pruning  |         |
|               |        | +MST           | "Best"  | Average    | "Best" | Ave NW  | Average         | Best    |
| Geometric     | 100    | -.21           | -13.88% | +18.5%     | -11.4% | +395.5% | -.02%           | -13.88% |
|               | 400    | -.28           | -4.07%  | +14.0%     | +8.8%  | +424.2% | -.09%           | -4.16%  |
|               | 1,600  | -.36           | -1.63%  | +13.2%     | +8.3%  | +683.2% | -.07%           | -1.42%  |
|               | 6,400  | -.40           | -.45%   | +10.3%     | +9.6%  | +245.0% | -.09%           | -.32%   |
|               | 25,600 | -.43           | -.17%   | +10.2%     | +8.6%  | +217.2% | -.02%           | -.09%   |
| Un-Structured | 100    | -1.03          | -5.84%  | +3.2%      | -4.2%  | +3.5%   | -.18%           | -3.67%  |
|               | 400    | -.89           | -1.60%  | +3.5%      | +1.0%  | +4.5%   | -.10%           | -1.09%  |
|               | 1,600  | -.89           | -.90%   | +4.5%      | +2.5%  | +5.4%   | -.07%           | -.63%   |
|               | 6,400  | -.88           | -.18%   | +4.0%      | +3.0%  | +4.7%   | -.07%           | -.13%   |
|               | 25,600 | -.94           | -.95%   | +4.0%      | +2.2%  | +4.9%   | -.10%           | -.95%   |

Table 2: Results for variants of the GW algorithm, expressed as percentage improvements (negative numbers) or degradations (positive numbers) compared to the average result for rooted strong pruning, over a selection of 100 different choices of root vertex. The “bests” are also over the 100 choices of root and so may not be true bests, except in the case of best pruning for unrooted growth, which yields the best over all roots. All columns except the “Ave NW” column concern the GW objective function; the latter concerns the New Worth objective function. The “+MST” column represents the average improvement obtained by using a minimum spanning tree algorithm for post-processing.

over 100 (essentially randomly chosen) roots, GW-pruning cannot do better than the *average* for strong pruning except on the smallest of the random graphs and one of the 12 street map instances. One *can* improve on the average strong pruning result by taking the best strong pruning result for the 100 roots, but the improvement is not substantial for the larger instances.

Thus, although GW-pruning has value as a proof technique, there would appear to be strong empirical support for using the simpler strong pruning approach in practice (although we shall have more to say about this in Section 4.2).

Table 2 also summarizes results for one other way of improving on strong pruning: Postprocess the tree by computing a minimum spanning tree (MST) on the vertices it contains. Note that there is no reason to believe that a GW-type algorithm will produce a minimum spanning tree, and our results confirm this. Typical improvements are 0.2% or more, with significantly larger improvements for the unstructured random graphs. The tables report only the improvement over average strong pruning results, but similar improvements apply to the

“best” results as well, at least on the larger instances, and for the other GW-algorithm variants. Since the time for computing an MST is negligible compared to that for the overall algorithm, this is definitely a worthwhile augmentation to the algorithm. We also investigated a postprocessing phase that took the non-zero prize vertices found by the algorithm and used the Goemans-Williamson Steiner Tree heuristic to construct a (hopefully) better Steiner tree for them, followed by MST postprocessing. Unfortunately, this was typically no better than doing the MST alone, and doubled the overall running time.

**3.5 Unrooted Growth: Theory and Experiments.** Our second proposed modification to the GW-algorithm concerns the growth phase. As defined, the GW-algorithm treats the root component as special. It does not get to use its surplus, even if it has one, to pay for edges. What if we instead use an *unrooted* growth phase, in which the root component is allowed to be active when it has a positive surplus? Theoretically, there is not much to be lost. With a slight modification of

| Type         | No. Instance | No. Vert. | No. Inst. | Without MST |          |         | With MST |          |         |
|--------------|--------------|-----------|-----------|-------------|----------|---------|----------|----------|---------|
|              |              |           |           | No. Opt.    | Ave. Gap | Max Gap | No. Opt. | Ave. Gap | Max Gap |
| Geometric    | 100          | 11        | 10        | .20%        | 2.25%    | 10      | .20%     | 2.25%    |         |
|              | 400          | 11        | 3         | 1.04%       | 3.35%    | 4       | .90%     | 3.35%    |         |
| Unstructured | 100          | 5         | 0         | 3.60%       | 6.98%    | 0       | 2.97%    | 6.94%    |         |
|              | 400          | 5         | 0         | 3.03%       | 4.45%    | 0       | 2.14%    | 4.21%    |         |
| OR-Lib       | 500          | 34        | 7         | 4.65%       | 18.18%   | 7       | 4.07%    | 18.18%   |         |
|              | 1000         | 29        | 7         | 4.12%       | 15.39%   | 8       | 3.80%    | 15.39%   |         |

Table 3: Comparison of solutions obtained by the GW-algorithm using unrooted growth and unrooted strong pruning to the optimal solutions for instances where the latter have been computed [8].

the original Goemans-Williamson proof to take into account the existence of one additional active component, we can prove the following analogue of the  $2 - \frac{1}{n-1}$  theorem for the original GW-algorithm:

**THEOREM 3.2.** *The algorithm obtained from the GW-algorithm by using an unrooted growth phase will always produce a tree  $T'$  with  $GW(T') \leq (2 - \frac{1}{n}) OPT$ .*

The unrooted growth-phase alternative has several advantages. The overall running time of the GW-algorithm is dominated by that for the growth phase, which is  $O(n^2 \log n)$  in both cases, versus  $O(n)$  for the pruning phase. Hence making the growth phase independent of the chosen root can yield substantial speedups if one wishes to construct trees for more than one choice of root. Furthermore, there is an  $O(n)$ -time recursive unrooted pruning technique that can be used with the unrooted growth phase to obtain the following:

**THEOREM 3.3.** *There is an  $O(n^2 \log n)$  running time  $(2 - \frac{1}{n})$ -approximation algorithm for the unrooted GW-minimization problem.*

To get a similar guarantee using the original GW-algorithm one might have to run the algorithm for all  $n$  potential roots, yielding a running-time bound of  $O(n^3 \log n)$ .

The above theorems unfortunately do not tell whether using unrooted growth with strong pruning will be better or worse in practice than using rooted growth. Based on the experimental results reported in Tables 1 and 2, the answer appears to be both. Let  $R(v)$  and  $U(v)$  be the GW values when the rooted/unrooted growth variants are applied for root  $v$ . For all three classes of instances, the average values for  $U(v)$  (over all roots) are typically better than the averages for  $R(v)$ , although only slightly (less than 0.1%). On the other hand, the best value of  $R(v)$  over a large selection of distinct roots is typically better (although again by less 0.1%) than the best value of  $U(v)$ , although the latter can be computed much more quickly using the algorithm of Theorem 3.3. Also note that if all we want is a solution for a specific root, the slight advantage in average

GW value for the unrooted growth version is often offset by a substantial disadvantage in running time, with the penalty being as much as a factor of 2, although typically being more like 1.5.

**3.6 Closeness to Optimality.** One question we cannot address completely is how close to optimal these algorithms are in practice. However, what limited evidence we do have is suggestive. Lucena and Resende have developed a branch-and-cut optimization algorithm for the unrooted cases of GW-minimization and NW-maximization [8], which they have successfully applied to all of our instances with 400 or fewer vertices (all of which are random graphs), as well as some variously structured 500- and 1000-vertex instances derived from Steiner Tree test cases in the OR-Library [3] by randomly assigning prizes to the required vertices. Here the appropriate point of comparison is GW-algorithm variant that uses unrooted growth and unrooted (i.e., “best”) strong pruning. Table 3 reports on the gaps found between this algorithm’s solutions and the optimal ones for those instances that Lucena and Resende have successfully solved. These results suggest that in practice the distance from optimality increases as instances get larger, although it leaves hope that the the average gap may be converging to something like 5%.

#### 4 Quotas, Budgets, and the Cost/Prize Tradeoff Curve

We begin with worst-case results. Although none of the algorithms involved are likely to be practical, the basic approach they introduce leads to algorithms that *are* cost-effective in practice, as we shall see in Section 4.2.

**4.1 Worst-Case Guarantees for the Quota and Budget problems.** A key first observation is that the Quota problem is a generalization of the well-studied  $k$ -MST problem, in which, given an edge-weighted graph  $G$  and an integer  $k$ , one wishes to construct the minimum cost subtree of  $G$  that contains at least  $k$  vertices. The latter is simply the Quota PCST with all prizes equal to 1 and quota equal to  $k$ .

Note that any polynomial-time  $\alpha$ -approximation algorithm for the (rooted/unrooted)  $k$ -MST problem yields a pseudopolynomial-time  $\alpha$ -approximation algorithm for the corresponding Quota PCST: simply model a prize of  $p$  at vertex  $v$  by a halo of  $2pn$  new vertices, all linked by 0-cost edges to  $v$ , and set  $k$  equal to  $2n$  times the original quota  $Q$ . (The  $2n$  multiplier is needed so that we can adequately handle 0-prize vertices.) Moreover, for the  $k$ -MST algorithms in the literature, this pseudopolynomial time bound can typically be reduced to polynomial time, since the halos of zero-cost edges can usually be handled implicitly by simple modifications to the algorithm. For example, if one of the steps of the algorithm is to apply the GW-algorithm (as it is in [2, 5]), we know that the first step of the growth phase will be to merge all components linked by zero-cost edges, and so we can treat a vertex and its halo as a single unit with surplus  $2pn + 1$ .

In this way it is easy to see that Garg’s “factor-of-5” and “factor-of-3  $k$ -MST algorithms [5] yield a polynomial-time algorithms for the Quota PCST that have the same guarantee, as does the “factor-of-2.5” algorithm for the unrooted  $k$ -MST of [2], which uses Garg’s algorithm as a subroutine.

What of the Budget PCST? We know of no previous work on this problem, but have obtained the following:

**THEOREM 4.1.** *If there is a polynomial-time 3-approximation algorithm for the (rooted or unrooted)  $k$ -MST problem, then there is a polynomial-time  $(5 + \epsilon)$ -approximation algorithm for the unrooted Budget PCST for any  $\epsilon > 0$ .*

The algorithm works by repeatedly running the Quota algorithm, using binary search to find a quota  $Q$  such that the tree produced for quota  $Q$  has total edge cost no more than the budget  $B$  and the tree produced for quota  $(1 + \epsilon)Q$  has edge cost exceeding  $B$ . We then output the tree for  $Q$ . To prove the theorem, we use the following separator lemma.

**LEMMA 4.1.** *Suppose  $T$  is a vertex-weighted tree in which no vertex has more than  $1/5$  the total weight. Then  $T$  can be split into 3 edge-disjoint subtrees, each of which contains at least  $1/5$  of the total weight (where a vertex that is shared between subtrees contributes its full weight to each of the subtrees that contains it).*

In order to get a better worst-case bound for the Budget problem by the above technique, we will need an algorithm for the Quota problem with a guarantee of 2 or less, in which case the Budget guarantee will drop from 5 to 3.

**4.2 Experimental Results and the Cost/Prize Tradeoff Curve.** As remarked above, we probably will

never want to use any of the above algorithms directly in practice. The prize-multiplier approach inherent in Garg’s “factor-of-3” algorithm can, however, be adapted to yield quite practical heuristics for the Quota and Budget problems.

Garg’s idea was to run the GW-algorithm on a series of PCST instances, each obtained from the original instance  $I$  by multiplying all prize values by a fixed *prize multiplier*  $\alpha$ . Let  $I_\alpha$  be the instance derived using multiplier  $\alpha$ . Consider the quota problem. Given a Quota  $Q$  and a small fixed  $\epsilon > 0$ , one uses binary search to find a value of  $\alpha$  such that the tree generated by the GW-algorithm for  $I_\alpha$  has total prize  $Q$  or more (as measured in terms of the original instance  $I$ ), but the tree generated for  $I_{\alpha/(1+\epsilon)}$  has total prize less than  $Q$ . One then either returns the first tree or, if it has significant excess prize, tries to find a better tree intermediate between the two trees.

In trying to understand the performance of this approach, a natural object to study is the “tradeoff curve” obtained by plotting the  $\langle \text{prize}, \text{cost} \rangle$  pairs (original, not multiplied prizes) obtained for various  $\alpha$ . Figure 1 illustrates such a curve for one of our street map instances. This curve was obtained using the rooted growth phase plus strong pruning, with  $\alpha$  increasing from 0.2 to 4.0 by factors of 1.001. (In the remainder of this section, we concentrate for specificity on the rooted version of the problem.) Note that no trees were found between the empty one consisting only of the 0-prize root, and the one that contains 28.7% of the total prize. Such an initial gap is a common phenomenon for instances

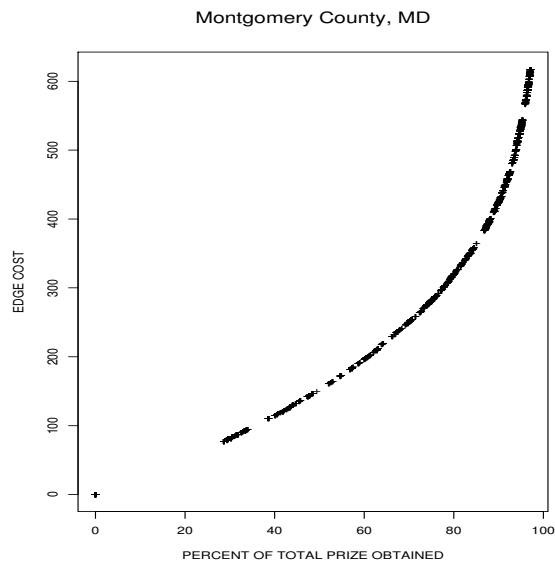


Figure 1: Tradeoffs between cost and prize obtained by varying the GW prize-multiplier.

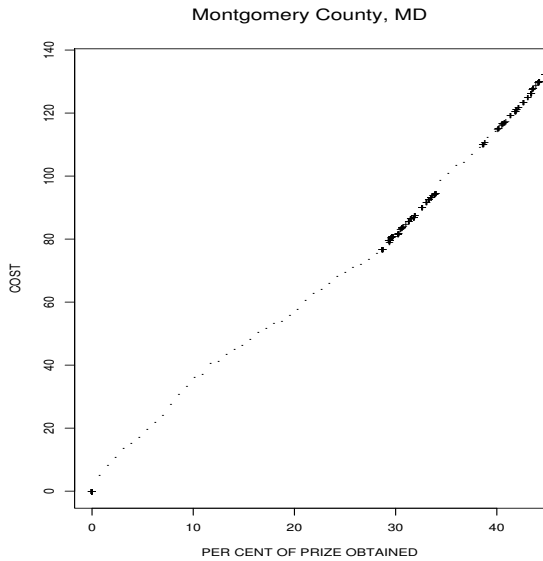


Figure 2: Interpolating into the prize/cost tradeoff curve of Figure 1 (“+” symbols) using greedy pruning (“.” symbols). Note that for each gap in the original curve, the interpolated points do not rise substantially above the straight lines linking the data points at the top and bottom of the gap.

like ours with most vertices having prize 0, but note that unless our quota was in that gap, we would likely be happy to settle for the tree of lowest cost that contains enough prize, without resorting to the complicated heuristic Garg invokes to find an intermediate tree.

For big gaps like the initial one, there is also an appealingly simple alternative: take the tree at the top end of the gap and find a minimal subtree that contains the root and enough prize to meet the quota. There exists a dynamic-programming-based approximation scheme to do this (which becomes an exact algorithm in the case of the  $k$ -MST), although this is likely to be prohibitively expensive in practice. Fortunately, it appears that one can get quite satisfactory results by the simple greedy expedient of repeatedly pruning away the feasible subtree with maximum ratio of cost to prize (where a subtree is *feasible* if removing it does not reduce the total prize below the quota). Figure 2 shows the result of using this approach for multiple quotas so as to fill in the early gaps in the tradeoff curve of Figure 1. Even better results can often be obtained for a given quota by greedily pruning from several different trees above the gap, and taking the best result. One also gets reasonable results even if one generates the initial tradeoff curve using a substantially larger growth factor for  $\alpha$  than the factor of 1.001 used here (thus computing fewer points and saving considerable computation time). An analogous pruning approach also appears to work well for the Budget problem.

Note that, for our application, the tradeoff curve itself may well be of more value than the solution to any particular instance of the Budget or Quota problem, since network planning typically has to optimize over a range of possibilities. Moreover, the curve turns out to be a fascinating subject of study in its own right, providing many surprises. For instance, although Figure 1 provides what appears to be a nicely monotonic curve, this is not always the case: for some instances, certain values of  $\alpha$  can produce trees that are totally dominated by others, i.e. have less prize and yet cost more. Indeed, a closer inspection of the tradeoff curve of Figure 1 shows that even it is riddled with such nonmonotonocities. See Figure 3.

Other nonmonotonocities are revealed by our data as well. For instance, in the process by which tradeoff curves are generated, increasing the prize multiplier  $\alpha$  may actually cause the prize obtained to decrease, a phenomenon that is quite noticeable under strong pruning and occasionally present under GW-pruning. Note that although most of the nonmonotonocities are due to the pruning process, this is not the only possibility. For rooted growth there are simple examples where increasing the multiplier reduces the number of vertices in the root component at the end of the growth phase. (For unrooted growth, however, this is impossible.)

**4.3 Comparing Tradeoff Curves.** For a given multiplier  $\alpha$ , strong pruning typically results in significantly less prize than does GW-pruning (as well as substantially less edge cost.) Thus it is hard to guess, *a priori*,

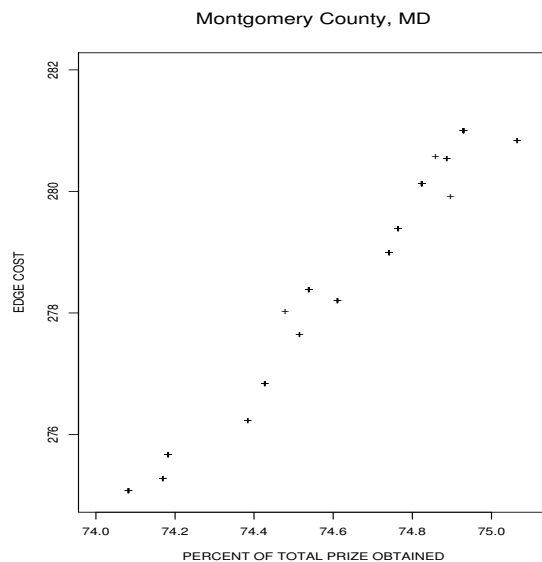


Figure 3: Closeup of the tradeoff curve of Figure 1 (“+” symbols) showing nonmonotonocities.

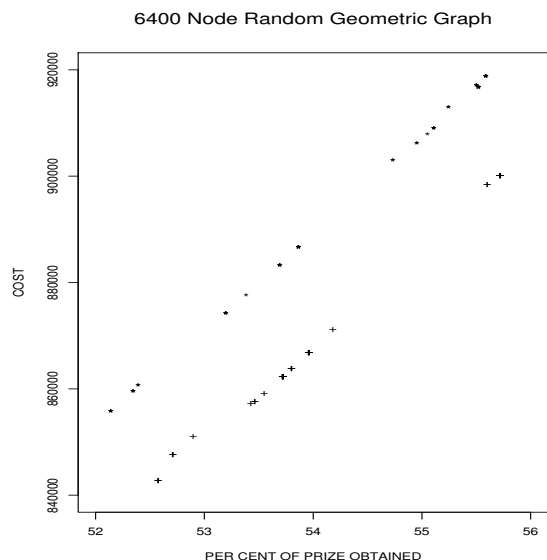


Figure 4: Close up of the tradeoff curves under strong pruning (“\*” symbols) and GW pruning (“+” symbols). Note that GW-pruning provides a distinctly better tradeoff curve than does strong pruning.

whether the two methods yield the same curves or, if not, which one yields the better curve. Surprisingly, GW-pruning typically turns out to be the winner, despite the fact that it is handily outperformed by strong pruning on the GW-minimization problem for which it was originally designed. (In GW-pruning’s favor, we can show, by a generalization of Claim 2.1 in [5] to the case of 0-1 prizes, that for each point on the GW-pruning tradeoff curve, the cost is no more than a factor of 2 greater than the best possible cost needed to obtain the given prize, a result that may well not hold for strong pruning.) Figure 4 provides a closeup of the two curves for one of our random geometric graphs, where the advantage of GW-pruning is particularly clear.

In order to confirm the existence of a widespread GW-pruning advantage, we need a way of making quantitative comparisons between tradeoff curves. Note that our “curves” are really just sets  $S$  of solutions. For a given set of solutions  $S$  and an  $x \in [0, 1]$ , let  $f_S(x)$  be the minimum cost of a solution in  $S$  that contains at least  $x$  times the total prize. Note that this makes  $f_S$  a step function. A closeup of the step function for strong pruning and Montgomery County is illustrated in Figure 5. Suppose we make a request for a random amount of prize, the amount chosen uniformly between 0 and the total available prize. Then the expected cost  $E[S]$  that we will pay, given that we are using  $S$  for our tradeoff curve, is  $E[S] = \int_0^1 f_S(x)dx$ , and this seems a plausible figure of merit for such a curve. Note that filling the gaps in the curve using the pruning heuristic

of the previous section should lower  $E[S]$ , as should generating more points in the first place using a smaller growth factor  $1 + \epsilon$ .

Let  $S_{GW}^F$  and  $S_{SP}^F$  be the sets of tradeoff points generated by GW-pruning and strong pruning when the growth factor is  $F$ . Let  $S_{GW'}^F$  and  $S_{SP'}^F$  be the corresponding “interpolated” pointsets obtained by using our pruning heuristic to fill all gaps larger than 1% of the total prize. For our streetmap instances, we used  $F = 1.1$  and had  $E[S_{GW}^F] < E[S_{SP}^F]$  for 11 of the 12 instances, with an average gap of 2.6%. For 10 of the 12, we had  $E[S_{GW'}^F] < E[S_{SP'}^F]$ , with an average gap of 2.1%. The average improvement of  $E[S_{GW'}^F]$  over  $E[S_{GW}^F]$  was 4.8%.

For 8 of the 11 random geometric instances of size 1600, we have  $E[S_{GW}^F] < E[S_{SP}^F]$  both for  $F = 1.1$  and  $F = 1.01$ , with the average gap being 2.3% in favor of GW-pruning in the first case, and 1.9% in the second. For the interpolated tradeoff curves, GW-pruning was better for all 11 instances and both choices of  $F$ , the average gap being 1.6% in the first case and 1.8% in the second. Because of gaps in the curves, the improvements in  $E[S_{GW}^F]$  due to interpolation were substantial: roughly 14.7% for  $F = 1.1$ , and 9.3% for  $F = 1.01$ . The overall improvement in  $E[S_{GW}^F]$  due to reducing the growth factor  $F$  to 1.01 and using interpolation was however only .9% more than that for using the  $F = 1.1$  and interpolation alone.

GW-pruning was also better on all 5 of our 6400-vertex geometric graphs, both before and after interpolation. Only with the unstructured random instances is

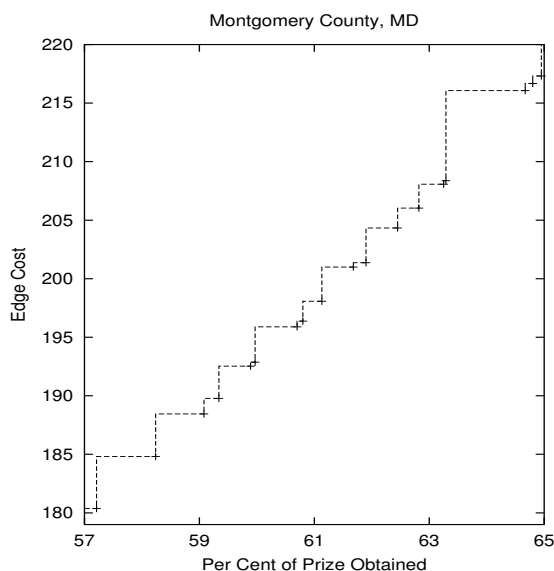


Figure 5: The step function.

the picture unclear: for the 1600- and 6400-vertex instances of this type, the two pruning techniques were roughly equivalent. A more complete study of this issue will appear in [9], along with expanded coverage of all the issues in this paper.

## 5 Conclusion.

We conclude with a brief discussion of the potential for further improvements on the GW-algorithm, in the context of the GW objective function, the first of which was derived from the results of the previous section. In studying the effect of varying the prize multiplier  $\alpha$ , we observed that for both pruning rules, the best trees with respect to the original GW-minimization problem are often obtained by using multipliers less than 1, i.e., by tricking the GW-algorithm into thinking it has less prize than it actually does, e.g., see Figure 6. This is common for GW-pruning, which as we have already seen leaves much room for improvement when the multiplier is 1. Here values of  $\alpha$  as small as .5 on occasion yield the best results. For strong pruning, the effect is much less pronounced, although for the unstructured random instance in Figure 6, we did get an improvement of 2.7% by taking  $\alpha = 0.795$  for our default choice of root.

To find such improvements, one must run the GW-algorithm many times. A perhaps more promising way

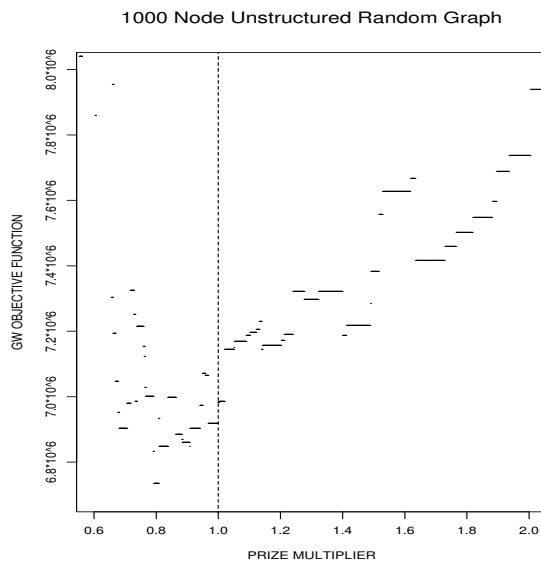


Figure 6: Value of GW-objective function (using original prize values) as a function of prize multiplier for rooted strong pruning and a random graph with edge probability 0.1, all edge weights equal to 100,000, and nonzero prizes on 10% of the vertices, with prize values distributed uniformly between 0 and 200,000. The best choice for the prize multiplier is about  $\alpha = .795$ , yielding a value for the GW objective function that is 2.7% better than that for  $\alpha = 1$ .

to exploit multiple runs is in the context of multiple-start local optimization, as is currently being explored by Canuto et al. [1]. Here one repeatedly runs the GW-algorithm on minor random perturbations to the instance, using the results as starting points for a sophisticated local improvement algorithm. On the instances with known optima covered in Table 3, a “best-of-500-starts” version of this algorithm averages less than 1% above optimal for each of the classes of problems covered [10]. Simply running the local improvement algorithm on the output of one run of the GW-algorithm also yields measurable (but significantly smaller) improvements. Presumably these approaches would result in better tradeoff curves as well, assuming one is able to spend the extra time, but we leave for future research the task of quantifying this effect.

**Acknowledgments.** Thanks to Sanjeev Arora, Naveen Garg, Howard Karloff, Abilio Lucena, Mauricio Resende, and David Williamson for helpful discussions.

## References

- [1] S. A. Canuto and C. C. Ribeiro and M. G. C. Resende. Local search with perturbations for the prize-collecting steiner tree problem. In *Proc. Third Metaheuristics International Conference (Extended Abstracts)*, pages 115–119. Catholic University of Rio de Janeiro, 1999.
- [2] S. Arya and H. Ramesh. A 2.5 approximation algorithm for the  $k$ -mst problem. *Information Processing Lett.*, 65:117–118, 1998.
- [3] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *J. Oper. Res. Soc.*, 41:1069–1072, 1990. Web page: <http://mscmga.ms.ic.ac.uk/info.html>.
- [4] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions, 1999. To appear.
- [5] N. Garg. A 3 factor approximation algorithm for the minimum tree spanning  $k$  vertices. In *Proc. 37th IEEE Symp. on Foundations of Computer Science*, pages 302–309, 1996.
- [6] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24:296–317, 1995.
- [7] M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, pages 144–191. PWS Publishing Company, Boston, 1997.
- [8] A. Lucena and M. G. C. Resende, 1999. Personal communication.
- [9] M. Minkoff. The prize collecting steiner tree problem. M.S. Thesis, Massachusetts Institute of Technology, in preparation.
- [10] M. G. C. Resende, 1999. Personal communication.