# The Constrained Minimum Spanning Tree Problem

(Extended Abstract)

R. Ravi[*] M. X. Goemans[†]

**Abstract**

Given an undirected graph with two different nonnegative costs associated with every edge $e$ (say, $w_e$ for the weight and $l_e$ for the length of edge $e$) and a budget $L$, consider the problem of finding a spanning tree of total edge length at most $L$ and minimum total weight under this restriction. This *constrained minimum spanning tree problem* is weakly NP-hard. We present a polynomial-time approximation scheme for this problem. This algorithm always produces a spanning tree of total length at most $(1+\epsilon)L$ and of total weight at most that of any spanning tree of total length at most $L$, for any fixed $\epsilon > 0$. The algorithm uses Lagrangean relaxation, and exploits adjacency relations for matroids.

**Keywords**: Approximation algorithm, minimum spanning trees, Lagrangean relaxation, adjacency relations.

## 1 Introduction

Given an undirected graph $G = (V, E)$ and nonnegative integers $l_e$ and $w_e$ for each edge $e \in E$, we consider the problem of finding a spanning tree that has low total cost with respect to *both* the cost functions $l$ and $w$. For convenience, we will refer to $l_e$ and $w_e$ of an edge $e$ as its length and weight respectively. Thus the problem we consider is that of finding a spanning tree with small total weight and small total length.

[*]GSIA, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh PA 15213. Email: `ravi+@cmu.edu`.

[†]MIT, Department of Mathematics, Room 2-382, Cambridge, MA 02139. Email: `goemans@math.mit.edu`. Research supported in part by NSF contract 9302476-CCR, ARPA Contract N00014-95-1-1246, and a Sloan fellowship.

This is a bicriteria problem. A natural way to formulate such problems is to specify a budget on one of the cost functions and minimize the other objective under this constraint. The problem therefore becomes a capacitated problem. In this case, we can specify a budget $L$ on the total length of the spanning tree and require a tree of minimum weight under this budget restriction. We call this problem the *Constrained Minimum Spanning Tree problem.*

**Lemma 1.1** [1] The constrained minimum spanning tree problem is (weakly) NP-hard.

Define an $(\alpha, \beta)$-approximation for this problem as a polynomial-time algorithm that always outputs a spanning tree with total length at most $\alpha L$ and of total weight at most $\beta W$, where $W$ is the minimum weight of any spanning tree of $G$ of length at most $L$. In other words, $W$ is the answer to the constrained minimum spanning tree problem formulated in the previous paragraph. Observe that the definition is not completely symmetric in the two cost functions; the quantity $L$ is given.

In this extended abstract, we first present a $(2,1)$-approximation algorithm for the constrained minimum spanning tree problem. The algorithm is based on Lagrangean relaxation, and the proof of the performance guarantee exploits the fact that two adjacent spanning trees on the spanning tree polytope differ by exactly two edges (one in each tree). Moreover, this algorithm can be implemented in almost linear time using an elegant technique of Meggido [6].

We then refine the algorithm to derive an approximation scheme. The precise result is given below.

**Theorem 1.2** For any fixed $\epsilon > 0$, there is a $(1+\epsilon, 1)$-approximation algorithm for the constrained minimum spanning tree problem that runs in polynomial time.

The same result holds if we replace the set of spanning trees by the bases of any matroid.

Note also that the above approximation can be used to derive a $(1, 1+\epsilon)$-approximation algorithm for the constrained minimum spanning tree problem that runs in pseudopolynomial time. This observation follows from more general arguments in [5]; we reproduce it here for completeness. In this latter problem, we must find a tree of length at most the budget $L$ and of cost at most $(1+\epsilon)$ times the minimum weight of any tree of length at most $L$. The idea is to use the weights rather than the lengths as the budgeted objective in the available algorithm. Consider running the given algorithm for all possible integral budget values on the weight of the tree to find a tree of approximately minimum length. Over all these runs, find the smallest value $W'$ of the budget such that the length of the tree output is at most $L$. Since no smaller value of the budget on the weight of the tree gives a tree of length at most $L$, it must be the case that $W'$ is a lower bound on the weight of any spanning tree of length at most $L$. But the tree obtained by running the given algorithm with a weight budget of $W'$ must have weight at most $(1+\epsilon)W'$, and therefore has the desired properties. Binary search can be used to speed up the determination of $W'$ using $O(\log W_{max})$ invocations of the given approximation algorithm, where $W_{max}$ is the sum of the largest $n-1$ weights.

## Related work

Aggarwal, Aneja and Nair [1] studied the constrained minimum spanning tree problem; they prove weak NP-hardness and describe computational experience with an approach for exact solution. Guignard and Rosenwein [3] apply a special form of Lagrangean relaxation to solve to optimality the directed version of the problem we consider here, that of finding constrained minimum arborescences.

There have not been too many approximation algorithms for bicriteria problems. This may come from the fact that capacitated problems are typically much harder than their uncapacitated counterparts. We mention here some work that is closely related; see also [9] for additional references. Lin and Vitter [4] provided approximations for the $s$-median problem where $s$ median nodes must be chosen so as to minimize the sum of the distances from each vertex to its nearest median. The solution output is approximate in terms of both the number of median-nodes used and the sum of the distances from each vertex to the nearest median. Shmoys and Tardos [10] studied the problem of scheduling unrelated parallel machines with costs associated with processing a job on a given machine. Given a budget on the cost of the schedule, they presented an approximation algorithm for minimizing the makespan of the schedule. Both the papers mentioned above use a linear programming formulation of the respective problems and use different rounding methods to round a fractional solution to a feasible integral solution. Even though these methods employ linear programming, our approach is quite different.

Recently, Marathe, R. Ravi, Sundaram, S.S. Ravi, Rosenkrantz and

Hunt studied several bicriteria network design problems in [5]. They presented a $(2,2)$-approximation algorithm for the constrained minimum spanning tree problem using a parametric search method combined with a cost-scaling technique. Their method also yields approximation algorithms for several bicriteria problems for which the two criteria are similar, i.e. both the objectives are of the same type but only differ in the cost function based on which they are computed. The constrained minimum spanning tree problem is such an example. Theorem 1.2 is an improvement of the result in [5] in two ways: the performance ratio is better, and the algorithm we present is strongly polynomial and does not depend on the magnitude of the costs assigned to edges. The method in [5] uses a cost-scaling approach and hence the running time depends on the magnitudes of the costs.

In the next section, we review Lagrangean relaxation as applied to our problem. Then we present the approximation algorithm in Section 3, and describe a fast implementation in Section 4.

## 2 Lagrangean relaxation

Lagrangean relaxation is a classical technique to get rid of a set of "hard" constraints in an optimization problem. This gives lower bounds (for minimization problems) on the optimum value. We refer the reader to Nemhauser and Wolsey [7] for a discussion of the method. In this section, we consider the application of Lagrangean relaxation to the constrained minimum spanning tree problem. In a subsequent section, we will show how to derive from this Lagrangean relaxation a spanning tree of approximately minimum length and weight.

Given a graph $G = (V, E)$, let $S$ denote the set of incidence vectors of spanning trees of $G$. The constrained minimum spanning tree problem can be formulated by the following optimization problem:

$$\begin{array}{lrcl} & W & = & \text{Min} \sum_{e\in E} w_e x_e \\ & & \text{subject to:} & \\ (IP) & & & x \in S \\ & & & \sum_{e\in E} l_e x_e \le L. \qquad (1) \end{array}$$

Considering the budget constraint (1) as the complicating constraint, we can obtain a lower bound on the optimum value $W$ by dualizing it and considering for any $z \geq 0$ the following minimum spanning tree problem:

$$\begin{array}{lrll} & l(z) & = & \text{Min} \quad \sum_{e \in E} (w_e + zl_e)x_e - zL \\ & & \text{subject to:} & \\ (P_z) & & & x \in S. \end{array}$$

The value $l(z)$ is clearly a lower bound on $W$ since any spanning tree which satisfies the budget constraint would give an objective function value in $(P_z)$ no higher than in $(IP)$. We observe that $(P_z)$ is simply a minimum spanning tree problem with respect to the costs $c_e = w_e + zl_e$. In order to get the best lower bound on $W$, we can maximize $l(z)$ over all $z \geq 0$ to obtain:

$$LR = \text{Max}_{z \geq 0}\ l(z).$$

We let $z^*$ denote the value of $z$ which maximizes $l(z)$, and let $c_e^* = w_e + z^* l_e$. It is well-known and easy to see that $l(z)$ is concave and piecewise linear. For an illustration, see Figure 1.

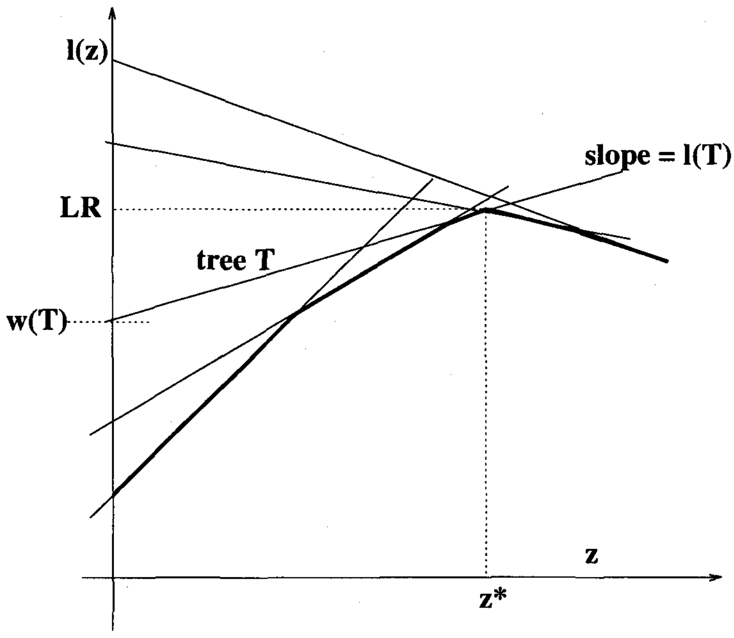

Figure 1: The plot of $l(z)$ as $z$ varies. Every spanning tree of weight $w(T)$ and length $l(T)$ corresponds to a line with intercept $w(T)$ and slope $l(T) - L$. The plot above is the lower envelope of the lines corresponding to all spanning trees in the graph.

## 3 The approximation algorithm

Our algorithm is based on solving the Lagrangean relaxation and deriving a good spanning tree out of it. Our main result will follow from the following theorem.

**Theorem 3.1** Let $\mathcal{O}$ denote the set of spanning trees of minimum cost with respect to $c^*$. There exists a spanning tree $T \in \mathcal{O}$ of weight at most $LR \leq W$ and of length less than $L + l_{max}$ where $l_{max} = \max_{e \in E} l_e$.

### Proof of Theorem 3.1:

The weight $w(T)$ of any tree $T$ in $\mathcal{O}$ is equal to

$$w(T) = [w(T) + zl(T) - zL] - z(l(T) - L) = LR - z(l(T) - L),$$

and therefore is at most $LR$ if and only if $l(T) \geq L$.

We start by establishing a well-known and simple property of $\mathcal{O}$. If we consider $z = z^* + \epsilon$ or $z = z^* - \epsilon$ for an arbitrarily small $\epsilon > 0$, the optimum spanning trees with respect to $w_e + zl_e$ must be contained in $\mathcal{O}$. This implies that $\mathcal{O}$ must contain a spanning tree $T_{\leq}$ of length at most $L$. If not, we would have that $l(z^* + \epsilon) > l(z^*)$, contradicting the optimality of $z^*$. Similarly, there must exist a spanning tree $T_{\geq}$ of length at least $L$ in $\mathcal{O}$.

To derive the existence of a tree in $\mathcal{O}$ of length between $L$ and $L + l_{max}$, we use the adjacency relationship on the spanning tree polytope (the convex hull of incidence vectors of spanning trees) given in the following lemma. This adjacency relationship follows from the fact that forests of a graph define a matroid, the graphic matroid.

**Lemma 3.2** The spanning trees $T$ and $T'$ are adjacent on the spanning tree polytope if and only if they differ by a single edge swap, i.e. there exist $e \in T$ and $e' \in T'$ such that $T - e = T' - e'$.

By considering the optimum face of the spanning tree polytope induced by the spanning trees in $\mathcal{O}$, this lemma implies that if we have two optimum spanning trees $T$ and $T'$ then there must exist a sequence $T = T_0, T_1, \ldots, T_k = T'$ of *optimum* spanning trees such that $T_i$ and $T_{i+1}$ are adjacent for $i = 0, \ldots, k-1$. If we take $T = T_{\leq}$ and $T' = T_{\geq}$, we derive that there must exist two *adjacent* spanning trees $T_i$ and $T_{i+1}$ both in $\mathcal{O}$ such that $l(T_i) \leq L$ and $l(T_{i+1}) \geq L$. But $T_i$ and $T_{i+1}$ differ only in one edge swap. Thus $l(T_{i+1}) = l(T_i) + l_{e_{i+1}} - l_{e_i} \leq l(T_i) + l_{max}$ where $e_i \in T_i - T_{i+1}$ and $e_{i+1} \in T_{i+1} - T_i$. This shows that $T_{i+1}$ has length at least $L$ and less than $L + l_{max}$, completing the proof.

### 3.1 High-level algorithm and its performance guarantee

Theorem 3.1 and its proof motivates the following algorithm. First of all, observe that we can assume without loss of generality that $l_e \leq L$ for all edges $e$ in $G$. Edges that have higher values of $l_e$ would never be included in a feasible solution and therefore can be discarded. Then, compute the value $z^*$ solving the Lagrangean relaxation. Among all optimum trees for the cost function $c_e = w_e + z^* l_e$, find one that satisfies the conditions of Theorem 3.1. Because we have pruned all edges with $l_e > L$, we have that $l_{max} \leq L$ and, as a result, the tree output has weight at most $LR \leq W$ and length at most $2L$. Therefore, this constitutes a $(2, 1)$-approximation algorithm, provided we can implement the various steps of the algorithm.

## 4 Implementation

In this section, we show that the $(2, 1)$-approximation algorithm can be implemented in almost linear time. In particular, we sketch an implementation that runs in time $O(m \log^2 n + n \log^3 n)$, where $m = |E|$ and $n = |V|$.

### 4.1 Solving the Lagrangean relaxation

In order to compute the value $z^*$, we use an algorithm of Meggido [6]. We briefly summarize his elegant method here.

For any value of $z$, we can compute a minimum spanning tree with respect to $c = w + zl$. We can also easily determine if $z < z^*$, $z = z^*$ or $z > z^*$. For this purpose, among all optimum trees with respect to $c$, we can find the two trees $T_{min}$ and $T_{max}$ which have smallest and largest length. This can be done by using a lexicographic ordering of the edges instead of the ordering induced by $c$; for example, to compute $T_{min}$, we use the ordering $(c_e, l_e) < (c_f, l_f)$ if $c_e < c_f$ or if $c_e = c_f$ and $l_e < l_f$. Then $z < z^*$ if $l(T_{min}) > L$, $z > z^*$ if $l(T_{max}) < L$, and $z$ is (a possible value for) $z^*$ otherwise.

Meggido's approach is the following. Suppose we try to find an optimum tree for the value $z^*$, without knowing $z^*$. For this purpose, we would like to sort the edges with respect to their costs at $z^*$. Given two edges $e$ and $f$, we can determine if $c_e^* < c_f^*$, $c_e^* = c_f^*$ or $c_e^* > c_f^*$ without actually knowing $z^*$! Indeed, we only need to determine the breakpoint, say $z_{ef}$, of the two linear functions $c_e$ and $c_f$ as a function of $z$ and determine if $z_{ef} < z^*$, $z_{ef} = z^*$ or $z_{ef} > z^*$. This can be done by two minimum spanning tree computations (to determine $T_{min}$ and $T_{max}$) at the value $z_{ef}$. Therefore, if we use an algorithm which makes $O(m \log m)$ comparisons to determine the ordering of the costs at $z^*$, we will be able to determine $z^*$ by $O(m \log m)$ minimum spanning tree computations.

However, Meggido proposed to do this much more efficiently. Instead of using a serial sorting algorithm, we can use a parallel sorting algorithm which takes say $O(\log m)$ rounds and in each round make $O(m \log m)$ comparisons [8]. The advantage is that in any round the $O(m \log m)$ breakpoints can be computed in advance, then sorted (this is not even necessary), and then one can use binary search to determine where $z^*$ lies compared to all these breakpoints by doing only $O(\log(m \log m)) = O(\log m)$ minimum spanning tree computations. Over all rounds, this algorithm therefore makes $O(\log^2 m)$ minimum spanning tree computations. If we use the fastest MST algorithm available [2], this results in a total running time of $O(m \log^2 n + n \log^3 n)$.

## 4.2 Finding a tree satisfying the conditions of Theorem 3.1

The second part of the implementation is how to find the tree whose existence is claimed in Theorem 3.1, once $z^*$ is known. The algorithm described in the previous section not only gives $z^*$ but also produces two trees $T_{min}$ and $T_{max}$ which are optimum for $z^*$ and such that $l(T_{min}) \leq L$ and $l(T_{max}) \geq L$.

Following the proof of Theorem 3.1, we need to compute a sequence of optimum trees $T_{min} = T_0, T_1, \cdots, T_k = T_{max}$ such that $T_i$ and $T_{i+1}$ are adjacent ($i = 0, \cdots, k-1$) and simply return the first tree of the sequence whose length is at least $L$. To compute the sequence, we simply repeatedly swap an edge $e$ in $T_{max}$ but not in the current tree with a maximum cost edge not in $T_{max}$ but on the cycle closed by $e$. This sequence will therefore end in $k = |T_{max} - T_{min}| \leq n - 1$ steps.

If we were to implement each swap naively, this would take $O(n)$ time per swap, for a total running time of $O(n^2)$. However, using dynamic trees [11], Sleator and Tarjan show how to make this series of $O(n)$ swaps in $O(n \log n)$ time.

Summarizing, the $(2, 1)$-approximation algorithm can be implemented in $O(m \log^2 n + n \log^3 n)$ time.

## 5 A polynomial-time approximation scheme

The (2, 1)-approximation algorithm can be turned into a PTAS by modifying the initial edge-pruning rule in the algorithm. Earlier, we pruned all edges with length greater then $L$ since no such edge would be used in any feasible solution. The approximation guarantee of $2L$ on the length of the solution then followed from Theorem 3.1. To reduce this ratio, we could prune away all edges whose length is greater than $\epsilon L$ for some fixed $\epsilon > 0$. Then $l_{max}$ would be at most $\epsilon L$, resulting in a final tree of length at most $(1 + \epsilon)L$. However, we may discard edges that could possibly be used in an optimal solution. The key observation is that at most $\frac{1}{\epsilon}$ of the pruned edges can be used in any optimal solution, and there are only $O(n^{O(\frac{1}{\epsilon})})$ choices of subsets of pruned edges that may occur in any optimal solution. For each one of these polynomially many choices, we include the chosen edges in the tree, shrink the connected components, and run our algorithm on the resulting graph with a budget value of $L$ minus the length of the chosen edges. The solution output is the tree with minimum weight among all the trees over all the choices (note that all these trees have length at most $(1 + \epsilon)L$). The proof that the weight of the tree output is at most the optimum value $W$ is completed by considering the running of the algorithm for the same choice of the chosen edges as in some optimal solution of the constrained minimum spanning tree problem. This completes the proof of Theorem 1.2.

**Acknowledgements** Thanks to Philip Klein and David Williamson for discussions that led to this line of work.

## References

[1] V. Aggarwal, Y. Aneja and K. Nair, "Minimal spanning tree subject to a side constraint," *Comput. Operations Res.* **9**, 287–296 (1982).

[2] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to algorithms*, McGraw Hill (1990).

[3] M. Guignard and M.B. Rosenwein, "An application of Lagrangean decomposition to the resource-constrained minimum weighted arborescence problem," *Networks* **20**, 345–359 (1990).

[4] J.-H. Lin and J.S. Vitter, "$\epsilon$-approximations with minimum packing constraint violation," *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, 771–782 (1992).

[5] M.V. Marathe, R. Ravi, R. Sundaram, S.S. Ravi, D.J. Rosenkrantz, and H.B. Hunt III, "Bicriteria network design problems," *Proc. of the 22nd ICALP*, LNCS 944, 487–498 (1995).

[6] N. Meggido, "Applying parallel computation algorithms in the design of serial algorithms," *Journal of the ACM* **30**, 852–865 (1983).

[7] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York (1988).

[8] F.P. Preparata, "New parallel-sorting schemes", *IEEE Trans. Comput.* **C-27**, 669–673 (1978).

[9] R. Ravi, M.V. Marathe, S.S. Ravi, D.J. Rosenkrantz, and H.B. Hunt III, "Many birds with one stone: Multi-objective approximation algorithms," *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, 438–447 (1993).

[10] D.B. Shmoys and E. Tardos, "Scheduling unrelated parallel machines with costs," *Proc., 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, 448–454 (1993).

[11] D.D. Sleator and R.E. Tarjan, "A Data Structure fo Dynamic Trees," *Journal of Computer and System Sciences* **26**, 362–391 (1983).