Chapter 27
Balancing Minimum Spanning and Shortest Path Trees

Samir Khuller* Balaji Raghavachari† Neal Young‡

Abstract

Efficient algorithms are known for computing a *minimum spanning tree*, or a *shortest path tree* (with a fixed vertex as the root). The weight of a shortest path tree can be much more than the weight of a minimum spanning tree. Conversely, the distance between the root and any vertex in a minimum spanning tree may be much more than the distance between the two vertices in the graph. Consider the problem of balancing between the two kinds of trees: Does every graph contain a tree that is "light" (at most a constant times heavier than the minimum spanning tree), such that the distance from the root to *any* vertex in the tree is no more than a constant times the true distance? This paper answers the question in the affirmative. It is shown that there is a continuous tradeoff between the two parameters. For every $\gamma > 0$, there is a tree in the graph whose total weight is at most $1 + \frac{\sqrt{2}}{\gamma}$ times the weight of a minimum spanning tree, such that the distance in the tree between the root and any vertex is at most $1 + \sqrt{2}\gamma$ times the true distance. Efficient sequential and parallel algorithms achieving these factors are provided. The algorithms are shown to be optimal in two ways. First, it is shown that no algorithm can achieve better factors in all graphs, because there are graphs that do not have better trees. Second, it is shown that even on a per-graph basis, finding trees that achieve better factors is NP-hard.

* Computer Science Department, University of Maryland, College Park, MD 20742. E-mail : samir@cs.umd.edu. This work was done while this author was at UMIACS and was supported by NSF grants CCR-8906949, CCR-9103135 and CCR-9111348.

† Computer Science Department, Pennsylvania State University, University Park, PA 16802. E-mail : rbk@cs.psu.edu.

‡ University of Maryland Institute for Advanced Computer Studies, College Park, MD 20742. E-mail : young@umiacs.umd.edu. Supported by NSF grants CCR-8906949 and CCR-9111348.

1 Introduction

Let a graph $G = (V, E)$ represent a feasible communications network. An edge (a, b) denotes the feasibility of adding a link from site a to site b. Each edge e has a nonnegative weight $w(e)$, which represents the distance between the sites. The weight is a measure of the cost to add the link, and also of the time taken for a message to travel along it. The weight of a network is the sum of the weights of its edges. A *minimum spanning tree*, T_M, is a spanning subgraph of G whose weight is minimum, i.e., the cheapest network that will allow the sites to communicate. Fast algorithms for computing a minimum spanning tree (MST) in a graph are known [9, 10, 12, 15].

Assume that there is a root vertex r in the network from which many messages are sent. We would like the messages to be sent along short paths from the root to the vertices in the network, so that the messages reach their destinations quickly. A *shortest path tree*, T_S, is a tree rooted at r such that the distance between the root and any vertex u in T_S is exactly the same as the length of the shortest path between u and r in G. Dijkstra's algorithm can be used for computing T_S [8, 9].

It is possible that the weight of T_S is significantly more than the weight of T_M. In the example of Fig. 1, in the n-vertex graph, the shortest path tree has a weight of $\Omega(n^2)$, whereas the weight of an MST is only $O(n)$.

Conversely, certain vertices that are close to the root in a graph can be far away from r in T_M. In the example of Fig. 2, all edges on the cycle have a weight of 1, except the edge to the root that closes the cycle. The weight of the MST is $n - 1$, but the vertex that is at a distance of $1 + \epsilon$ in the graph is at a distance of $n - 1$ in T_M.

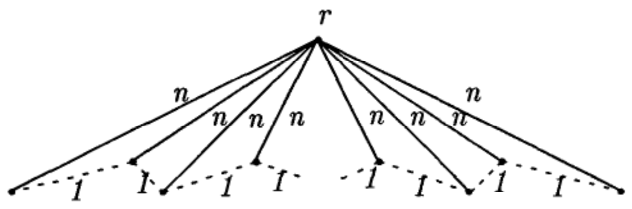

Figure 1: A Heavy Shortest-Path Tree

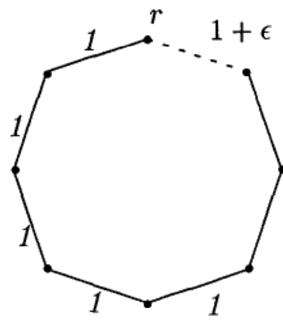

Figure 2: An MST with a Long Path

This raises the question of finding a spanning tree T whose total weight is not much more than the weight of T_M, and such that the distance from the root to any vertex in T is not much more than the distance in T_S. More precisely we ask the following question: Does every graph contain a tree that is "light" (at most a constant times heavier than the minimum spanning tree), such that the distance from the root to any vertex in the tree is no more than a constant times the true distance? We refer to such a tree as a *Light Approximate Shortest-path Tree* (LAST).

DEFINITION 1.1. *Let G be an arbitrary graph with non-negative edge weights and a root vertex r. A tree rooted at r is called an (α, β)-LAST if the following conditions are satisfied $(\alpha, \beta \geq 1)$:*

1. *The distance of every vertex v from r in T is at most α times the distance between v and r in G.*

2. *The weight of T is at most β times the weight of a MST of G.*

The main result in this paper, given in §3, is that for any $\alpha \geq 1$ and $\beta \geq 1 + \frac{2}{\alpha - 1}$, every graph contains an (α, β)-LAST for any root. Our proof is constructive: we give an algorithm that, given the n-vertex trees T_M and T_S, finds the (α, β)-LAST in $O(n)$ time.

In §4, we show that for $\alpha > 1$ this result is optimal: for any $\alpha > 1$ and $1 \leq \beta < 1 + \frac{2}{\alpha - 1}$, there are graphs containing no (α, β)-LAST at some root, and the problem of deciding whether a given graph contains an (α, β)-LAST is NP-complete.

For the remaining case, when $\alpha = 1$, the problem reduces to finding a shortest-path tree of at most a certain weight. In §5, we describe how the problem of finding a minimum-weight shortest-path tree can be reduced to the problem of finding a minimum-weight *branching* in the shortest-path subgraph of G, and how this yields an efficient algorithm for finding an optimal $(1, \beta)$-LAST.

In §6, we sketch how the algorithm from §3 can be parallelized to run in $O(\log n)$ time using n processors.

2 Related Work

Awerbuch, Baratz and Peleg [2] showed that every graph has a *shallow-light* tree, which is a tree that is at most a constant times heavier than T_M, and with the property that the diameter of T is at most a constant times the diameter of G. They also gave an algorithm to compute such a tree efficiently. For any graph, if we fix the root arbitrarily and fix α to be an arbitrary constant greater than one, then every $(\alpha, 1 + \frac{2}{\alpha - 1})$-LAST is also a shallow-light tree. Recently, we learned that Awerbuch, Baratz and Peleg [3] have extended their shallow-light-tree algorithm to obtain an $(\alpha, 1 + \frac{4}{\alpha - 1})$-LAST in $O(m + n \log n)$ time.

For VLSI applications of LAST's, see [5, 6].

Considerable research has been done on finding *spanners* of small size and weight in arbitrary graphs, and in Euclidean graphs (graphs induced by points in the plane). A t-spanner is a small weight spanning subgraph G' of G, with the property that the distance between any two vertices in G' is at most t times the distance in G. It is known that there are graphs that *do not* have spanners that are only a constant times heavier than T_M.

However, in our case we are looking for a "rooted" spanner. For a survey of results on spanners, see the recent paper by Chandra, Das, Narasimhan and Soares [4]. Other papers dealing with spanners are by [1, 13].

3 The Algorithm

Notation: Let $D_T(u,v)$ be the distance along the shortest path between vertices u and v in T. The weight of a subgraph X (such as a path or a tree) is denoted by $w(X)$ and is defined as the sum of the weights of the edges in X.

The idea is the following: we start with the tree T_M, and do a depth-first-search (DFS) of the tree, starting at the root. When we encounter a vertex for the first time, if its distance from the root in the current graph is too large, we bring it closer by adding its shortest path (in T_S) to the current graph.

The added paths are not too heavy because we do not add a path to a vertex unless the vertex is far from r in T_M, using any of the previously added paths. This allows us to bound the added weight by charging it to a path in T_M — if we add a path to a vertex v, and the last path added was to vertex u, then the length of the added path is bounded by $(D_{T_S}(r,u)+D_{T_M}(u,v))/\alpha$. By summing these bounds, we bound the net added weight. This method is inspired by the shallow-light tree algorithm of [2]. The fundamental differences are that we added paths to vertices that are too far from the root, rather than the last such vertex, and that we obtain an $O(n)$-time algorithm.

3.1 Relaxation. To obtain a simple linear-time algorithm, we don't add entire paths, and we don't keep track of exact shortest-path distances in the current graph. Instead, using standard shortest-path techniques [7, pp. 518], we maintain for each vertex v a *distance estimate* $d[v]$ and a *parent pointer* $p[v]$. The distance estimate of v is always an upper bound on the distance to r from v in the current graph via the path $(v, p[v], p[p[v]], ..., r)$. This invariant is established by the INITIALIZE step and maintained through a sequence of RELAX steps:

```
INITIALIZE()
Initialize distance estimates, parent pointers.
1  for $v \in V - \{r\}$ do $p[v] \leftarrow r$; $d[v] \leftarrow \infty$
2  $d[r] \leftarrow 0$
RELAX($u, v$)
Check for shorter path to $v$ through $u$.
1  if $d[v] > d[u] + w(u,v)$
2      then $d[v] \leftarrow d[u] + w(u,v)$
3           $p[v] \leftarrow u$
```

If during a sequence of relaxation steps, a path $P = (r = v_0, v_1, v_2, ..., v)$ has been relaxed (i.e., relaxations have been done in order on the pairs $\{(v_i, v_{i+1})\}$, possibly interspersed with other relaxations), then $d[v]$ is at most $w(P)$. Thus, for instance, if P is a shortest path, then $d[v]$ must equal $w(P)$, and the path $(v, p[v], p[p[v]], ..., r)$ must be a shortest path.

3.2 A Relaxing DFS. For our algorithm, we do a simple sequence of relaxation steps, and we use $d[v]$ to decide whether to add the shortest path to v. As we traverse T_M, we relax each edge (u,v) when we traverse it — once when we explore v from u (executing RELAX(u,v)), and later when we return from exploring v (executing RELAX(v,u)). When we first visit a vertex v, if $d[v] > \alpha\, D_{T_S}(r,v)$, then we "add the path" from r to v (in T_S) by relaxing enough edges on the path to lower $d[v]$ sufficiently.

When we finish the traversal, each distance estimate $d[v]$ is bounded by $\alpha\, D_{T_S}(r,v)$, and so the tree T formed by the edges $\{(v,p[v])\}$ meets the α constraint. If during the traversal, we add the shortest path to a vertex v, then we know that the weight of the path is bounded by $(D_{T_S}(r,u) + D_{T_M}(u,v))/\alpha$, because the corresponding path (from r to u via the edges in T_S, and then from u to v via the edges in T_M) was relaxed. Thus we can bound the weight of the added edges as described earlier.

The algorithm is given in Fig. 3.

3.3 Analysis of the Algorithm. We now prove that T has the desired properties: the distance from r to any vertex v in T is not increased by more than a factor of α, and $w(T)$ is no more than $\left(1+\frac{2}{\alpha-1}\right)w(T_M)$.

```
FIND-LAST($T_M, T_S, r, \alpha$)
Return an $(\alpha, 1+\frac{2}{\alpha-1})$-LAST.
1  INITIALIZE()
2  DFS($r$)
3  return tree $T = \{(v, p[v]) \mid v \in V - \{r\}\}$

DFS($u$)
Traverse the subtree of $T_M$ rooted at $u$, relaxing
edges as they are traversed, and adding paths from
$T_S$ as needed.
1  if $d[u] > \alpha\, D_{T_S}(r, u)$
2     then ADD-PATH($u$)
3  for each child $v$ of $u$ in $T_M$
4       do RELAX($u, v$)
5           DFS($v$)
6           RELAX($v, u$)

ADD-PATH($v$)
Relax edges along path from $r$ to $v$ in $T_S$.
1  if $d[v] > D_{T_S}(r, v)$
2     then ADD-PATH(parent$_{T_S}(v)$)
3            RELAX(parent$_{T_S}(v), v$)
```

Figure 3: Algorithm to Compute a LAST

LEMMA 3.1. *The distance of each vertex v from r in T is at most α times the distance in T_S, i.e.,*

$$D_T(r, v) \le \alpha\, D_{T_S}(r, v), \text{ for all } v \in V$$

Proof. Clearly the series of relaxation steps done by the algorithm guarantees that once a vertex v has been visited, $d[v] \le \alpha\, D_{T_S}(r, v)$. Subsequently, $d[v]$ does not increase, and, as described above, the weight of the path $(v, p[v], p[p[v]], ..., r)$ is always at most $d[v]$. When the algorithm terminates, this is the path from v to r in T. □

LEMMA 3.2. *The weight of T is at most $\left(1 + \frac{2}{\alpha-1}\right)$ times the weight of a minimum spanning tree, i.e.,*

$$w(T) \le \left(1 + \frac{2}{\alpha - 1}\right) w(T_M)$$

Proof. Let $v_0 = r$ and let $(v_1, v_2, ..., v_k)$ be the list of vertices that caused shortest paths to be added during the traversal, in the order they were encountered. When v_i is encountered, the weight of the newly added edges is at most $D_{T_S}(r, v_i)$. We will bound $\sum_i D_{T_S}(r, v_i)$ as follows.

Observe that, when v_i is encountered, $\alpha\, D_{T_S}(r, v_i) < d[v_i]$. As described earlier, the path P, composed of the path from r to v_{i-1} in T_S followed by the path from v_{i-1} to v_i in T_M, had been relaxed. Thus $d[v_i] \le w(P)$, and

$$\begin{aligned} \alpha\, D_{T_S}(r, v_i) &< d[v_i] \\ &\le w(P) \\ &= D_{T_S}(r, v_{i-1}) + D_{T_M}(v_{i-1}, v_i) \end{aligned}$$

Summing over i gives

$$\alpha \sum_{i=1}^{k} D_{T_S}(r, v_i) < \sum_{i=1}^{k} [D_{T_S}(r, v_{i-1}) + D_{T_M}(v_{i-1}, v_i)]$$

and therefore

$$(\alpha - 1) \sum_{i=1}^{k} D_{T_S}(r, v_i) < \sum_{i=1}^{k} D_{T_M}(v_{i-1}, v_i).$$

The DFS traversal traverses each edge exactly twice, and hence the sum on the right-hand side is at most twice the weight of T_M, i.e.,

$$\sum_{i=1}^{k} D_{T_M}(v_{i-1}, v_i) \le 2\, w(T_M).$$

Thus, if any paths are added, their net weight is less than $\frac{2}{\alpha-1} w(T_M)$. The remaining edges are from T_M, thus giving the result. □

THEOREM 3.1. (CORRECTNESS)
The algorithm finds a $(1 + \sqrt{2}\gamma, 1 + \frac{\sqrt{2}}{\gamma})$-LAST for any $\gamma > 0$.

Proof. Let $\alpha = 1 + \sqrt{2}\gamma$. Then by Lemmas 3.2 and 3.1 we get

$$\begin{aligned} w(T) &< \left(1 + \frac{\sqrt{2}}{\gamma}\right) w(T_M) \\ D_T(r, v) &\le \left(1 + \sqrt{2}\gamma\right) D_{T_S}(r, v) \end{aligned}$$

for all $v \in V$. Hence the tree generated by the algorithm is a $(1 + \sqrt{2}\gamma, 1 + \frac{\sqrt{2}}{\gamma})$-LAST. □

Notice that these also imply an upper bound on the diameter of T of $2(1+\sqrt{2}\gamma)$ times the diameter of G.

We now establish the running time of the algorithm.

THEOREM 3.2. (RUNNING TIME) *The algorithm runs in $O(n)$ time. Consequently, an (α, β)-LAST can be found in an arbitrary n-vertex, m-edge graph in $O(m + n \log n)$ time.*

Proof. The number of steps is proportional to the number of relaxations. Clearly at most $2n$ relaxations occur in DFS, and at most n occur in ADD-PATH. T_M and T_S can be found in $O(m + n \log n)$ time using existing algorithms [9, 10]. □

Observe that for Euclidean graphs (graphs induced by points in the plane) the running time is only $O(n \log n)$ since the MST can be computed in $O(n \log n)$ time [14]. The rest of the processing can be done in linear time.

Observation 1: Note that T_M and T_S can be arbitrary rooted trees — our algorithm will still compute a tree spanning the vertices of T_M of weight at most $\left(1 + \frac{2}{\alpha-1}\right) w(T_M)$, and in which the distance from r to each v is bounded by $\alpha\, D_{T_S}(r, v)$. Thus, if trees approximating the minimum weight and shortest paths are known or can be found quickly, a LAST can also be found quickly, although the weight and distance blow-up may increase by the additional approximation factors.

Observation 2: The algorithm can be used to solve the "multiple-root" problem, where there are multiple roots, and each vertex is required to be close to some root in T relative to its distance to its closest root in T_S.

4 Optimality of the Algorithm

In this section we show that the algorithm is optimal in the following sense. We show that for any $\alpha > 1$ and $1 \le \beta < 1 + \frac{2}{\alpha-1}$ there is a graph such that the graph does not contain an (α, β)-LAST, and that the problem of deciding whether a given graph contains an (α, β)-LAST is NP-complete.

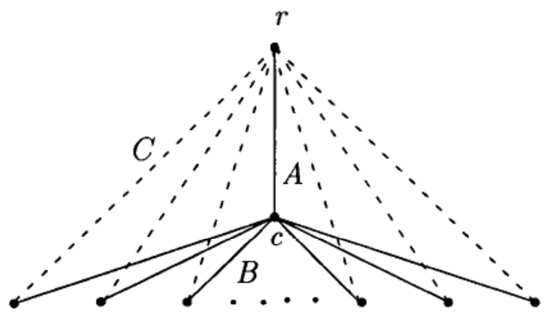

Figure 4: A graph with no (α, β)-LAST for $\beta < 1 + \frac{2}{\alpha-1}$. ($A = \alpha + 1$, $B = \alpha + \epsilon - 1$ and $C = 2$.)

4.1 Non-Existence of a LAST.

Consider the graph shown in Fig. 4. The structure of the graph is as follows. The root r is connected to a central vertex c by a path of weight A of edges of weight some small δ. The central vertex is connected through similar paths of weight B to the ℓ leaves. The root is connected to each leaf with an edge of weight C. Let $A = \alpha + 1$, $B = \alpha + \epsilon - 1$ and $C = 2$, where ϵ is an arbitrarily small constant. The MST is formed by using all edges except those of weight C.

Any (α, β)-LAST is forced to have all of the cost C edges. Since the length of the shortest path from the root to any leaf is 2, we cannot afford to add the path going through the center vertex (which is of length $A + B = 2\alpha + \epsilon$). The path to a leaf through some other leaf is of length $2 + 2B = 2(\alpha + \epsilon)$, which is also too long. Thus we are forced to add the edges from the root to every leaf. In addition, all but ℓ of the remaining edges are present. Therefore the weight of any (α, β)-LAST is $2\ell + T_M - \ell\delta$. More formally,

THEOREM 4.1. *For any $\alpha > 1$ and $\beta, 1 \le \beta < 1 + \frac{2}{(\alpha-1)}$, there exists a rooted graph containing no (α, β)-LAST.*

Proof. The minimum spanning tree T_M has a weight of $(\alpha + 1) + \ell(\alpha - 1 + \epsilon)$. Any (α, β)-LAST has weight $2\ell + T_M - \ell\delta$. The ratio of the weight of such a tree and T_M tends to $1 + \frac{2}{\alpha-1}$ as $\epsilon, \delta \to 0$ and $\ell \to \infty$. Hence, for every fixed $\beta < 1 + \frac{2}{\alpha-1}$, there is a graph that does not have an (α, β)-LAST. □

4.2 NP-Completeness of LAST Queries. We have shown that in general, no algorithm can guarantee anything better than an $(\alpha, 1+\frac{2}{\alpha-1})$-LAST, once α is fixed. Our proof was based on certain graphs having no such trees. An obvious question to ask is whether one can do better in graphs that do have the required tree as a subgraph. Suppose for fixed parameters α and β, with $\alpha > 1$ and $1 \leq \beta < 1+\frac{2}{\alpha-1}$, we ask whether a given graph has an (α,β)-LAST. We show that this problem is NP-complete.

THEOREM 4.2. *For any $\alpha > 1$, and $1 \leq \beta < 1+\frac{2}{\alpha-1}$, the following problem is NP-complete: "Given G, and a root r, does G contain an (α,β)-LAST?"*

Clearly, the problem is in NP. The proof of NP-hardness is in two parts. We first show that for any α, and $\beta = 1$, the problem is NP-complete. We then show how to do a reduction from the $(\alpha,1)$-LAST problem to the (α,β)-LAST problem for the specified range of β.

LEMMA 4.1. *The $(\alpha,1)$-LAST problem is NP-hard.*

Proof. We prove this by a reduction from 3-SAT. Let F be 3-SAT formula in conjunctive normal form — each clause is a subset of three literals from the sets $\{x_1,\ldots,x_n\}$ and $\{\overline{x}_1,\ldots,\overline{x}_n\}$. We build a graph in which the $(\alpha,1)$-LAST's correspond to satisfying assignments of F.

There will be a root vertex R. For each variable x_i we will build the following "gadget" to simulate the setting of a variable to be true or false. There is a "triangle" $S, X_i, \overline{X}_i$ corresponding to each variable x_i (see Fig. 5).

The weight of the edges (S, X_i) and $(S, \overline{X}_i)$ is A. There is a path connecting X_i and $\overline{X}_i$ of length E. There is also a path connecting R to S of length D. These paths are formed with small enough edges to ensure that the edges are in any MST.

For each clause c_j there is a vertex C_j, with an edge from R to C_j of weight W. If $x_i \in c_j$ then we add an edge from X_i to C_j of weight B. If $\overline{x}_i \in c_j$ then we add an edge from $\overline{X}_i$ to C_j of weight B.

Observe in this construction that, if $A < B < W$, the minimum spanning trees are exactly characterized by the following. The paths $[R,S]$ and $[X_i,\overline{X}_i]$ belong to the MST. For each variable x_i, exactly one of the two edges $\{(S,X_i),(S,\overline{X}_i)\}$ is in the MST. For each clause c_j, exactly one edge of the form (X_i,c_j) or $(\overline{X}_i,c_j)$ for some i is in the MST. No other edges are in the MST.

Next, we use the α constraint to ensure that the MST is an $(\alpha,1)$-LAST iff the path to each clause vertex comes from some variable vertex X_i or $\overline{X}_i$ that has an edge directly to S. If we can do this, then we are done — the $(\alpha,1)$-LAST's will correspond to satisfying assignments in the original formula via the following. For each variable x_i, choose the edge (S,X_i) iff x_i is true, otherwise choose the edge $(S,\overline{X}_i)$; for each clause c_j, choose the edge (X_i,c_j) (or $(\overline{X}_i,c_j)$), where x_i (or $\overline{x}_i$) is a variable (or negated variable) satisfying c_j.

To ensure this, it suffices that the weights A,B,D,E,W are picked so that they satisfy the following constraints:

$$A+D+E \leq \alpha \min\{A+D, B+W\}$$

$$A+B+D \leq \alpha \min\{A+B+D, W\} < A+B+D+E.$$

For instance, take $A = 1$, $B = \alpha$, $D = 2\alpha$, $E = (\alpha-1)(2\alpha+1)$, and $W = 1+2\alpha+\frac{1}{\alpha}$. □

We now provide the proof of Theorem 4.2.

Proof. We now reduce the $(\alpha,1)$-LAST problem to the (α,β)-LAST problem (for $\alpha > 1$, $1 \leq \beta < 1+\frac{2}{\alpha-1}$).

Let G^* be the graph for which we want to solve the $(\alpha,1)$-LAST problem. By Theorem 4.1, there exists a graph G' with no (α,β)-LAST. Assume without loss of generality that the MST of G^* has weight 1 and the MST of G' is of weight c (a constant to be determined later). Define the graph G to be the union of G^* and G' by identifying their roots into a single root r.

Let β' be the minimum such that G' has an (α,β')-LAST. Define β^* analogously for G^*. Take $c = \frac{\beta-1}{\beta'-\beta}$.

The weight of the MST in G is $1+c$, similarly the lightest tree in G meeting the α requirement is of weight $\beta^*+\beta' c$. Thus G has an (α,β)-LAST iff $\beta^*+\beta' c \leq \beta(1+c)$. By our choice of c, this is equivalent to $\beta^* \leq 1$. Thus G has an (α,β)-LAST iff G^* has an $(\alpha,1)$-LAST. □

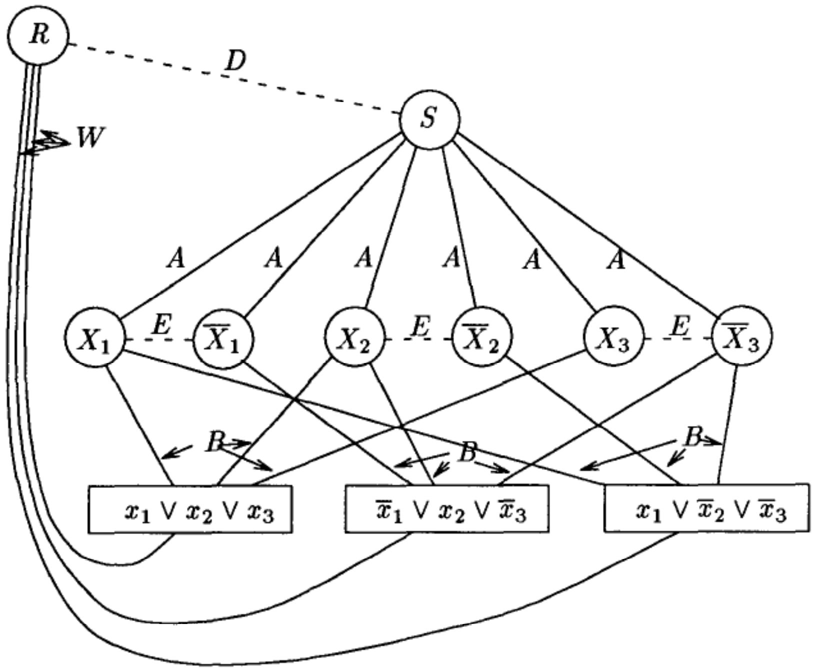

Figure 5: Reduction from 3-SAT

5 Minimum-Weight Shortest-Path Trees

Finally, we consider what happens when $\alpha = 1$. Here, an (α, β)-LAST is a shortest-path tree from the root, of weight at most β times the weight of the minimum spanning tree. In this section, we give an efficient method for finding the minimum-weight shortest-path tree.

The *shortest-path subgraph* in a rooted, weighted, directed graph is obtained from the graph by removing those edges (u, v) *not* on any shortest path from the root — those for which $D_{T_S}(r, u) + w(u, v) > D_{T_S}(r, v)$. It is easily shown that paths from r in the shortest-path subgraph correspond to shortest paths from r in the original graph, and vice versa.

A *branching* in a rooted, directed graph is a spanning tree with all edges directed away from the root. Efficient algorithms for finding minimum-weight branchings are known [10].

Thus, finding a minimum-weight shortest path tree in a directed graph reduces to finding a minimum-weight branching in the shortest-path subgraph. To find the minimum-weight shortest path tree in an undirected graph, simply direct it: duplicate each edge, directing one copy to each endpoint. Shortest-path trees in the original graph, directed appropriately, then correspond to shortest-path trees in the directed graph.

6 Parallelizing the LAST Algorithm

In this section we sketch how the LAST algorithm can be parallelized. The parallel algorithm runs in $O(\log n)$ time using n processors on a CREW PRAM. We give details in the full paper.

Briefly, replace each edge $\{u, v\}$ in T_M with the two directed edges (u, v) and (v, u). Let $C = (e_1, e_2, ..., e_{2n-2})$ be the edges of an Euler tour representing a DFS traversal of T_M, starting at the root. Let $e_i = (u_i, u_{i+1})$. Abusing notation, let $D_C(i, j)$ denote $\sum_{k=i}^{j-1} w(e_k)$. For $i < j$, define the relation $m(i, j)$ to be true if and only if

$$D_{T_S}(r, u_i) + D_C(i, j) > \alpha\, D_{T_S}(r, u_j).$$

For fixed j, the relation $m(i, j)$ is monotone in i. We show that this is sufficient to compute the function $M(i) = \min\{j > i : m(i, j)\}$ in $O(\log n)$

time and n processors. Intuitively, $u_{M(i)}$ is the next vertex that will have to have its shortest path added if u_i does.

We construct the LAST tree T as follows. Let $p_S(v)$ denote the parent of vertex v in T_S. Let $p_M(v)$ denote the parent of v in T_M. Let Q be the smallest set containing 0 and closed under $M(\cdot)$. Let R be the smallest set containing $\{u_i : i \in Q\}$ and closed under $p_S(\cdot)$. Assign $p[v] \leftarrow p_S(v)$ for $v \in R$, and $p[v] \leftarrow p_M(v)$ otherwise. Take the tree T to consist of the edges $\{(v, p[v])\}$.

7 Conclusions

We have demonstrated that every graph contains trees that offer a continuous tradeoff between minimum spanning trees and shortest path trees. Our proof is constructive and yields a simple and efficient algorithm.

As a corollary of our result, we can approximate the case when we need to compute light trees in which the sum of distances from the root to each vertex is minimum. The quality of the approximation is the same as in our problem. Is it possible to do better? What about a tree with summation of all pairs distances or with a fixed set of roots?

One could pose the same question for digraphs, and ask for the existence of a branching (with a root r), with the property that the distance from r to any vertex is at most a constant time the true distance, and the weight is not much more than the minimum weight branching. It is not difficult to see that there are graphs for which a branching with this property does not exist.

Acknowledgments

We would like to thank Shay Kutten for telling us about [2]. We would also like to thank Seffi Naor, Dheeraj Sanghi and Moti Yung for useful discussions.

References

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, Generating sparse spanners for weighted graphs, Proc. of 2nd Scandinavian Workshop on Algorithm Theory (SWAT), pp. 26–37, (1990).

[2] B. Awerbuch, A. Baratz, and D. Peleg, Cost-sensitive analysis of communication protocols, Proc. of 9th Symp. on Principles of Distributed Computing (PODC), pp. 177–187, (1990).

[3] B. Awerbuch, A. Baratz, and D. Peleg, Efficient broadcast and light-weight spanners, Manuscript, (1991).

[4] B. Chandra, G. Das, G. Narasimhan and J. Soares, New sparseness results on graph spanners, Proc. of 8th Symp. on Computational Geometry, (CG), pp. 192–201, (1992).

[5] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh and C. K. Wong, Performance-driven global routing for cell based IC's, Proc. IEEE Intl. Conference on Computer Design, pp. 170–173, (1991).

[6] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh and C. K. Wong, Provably good performance-driven global routing, *IEEE Transactions on CAD*, pp. 739-752, (1992).

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, *The MIT Press*, (1989)

[8] E. W. Dijkstra, A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik*, 1, pp. 269–271 (1959).

[9] M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *Journal of the ACM*, 34 (3), pp. 596–615, (1987).

[10] H. N. Gabow, Z. Galil, T. Spencer and R. E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, *Combinatorica*, 6 (2), pp. 109–122, (1986).

[11] J. JáJá, Introduction to Parallel Algorithms, *Addison-Wesley*, Reading, MA, (1991).

[12] J. B. Kruskal, On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, *Proc. Amer. Math. Soc.*, 7, pp. 48–50 (1956).

[13] D. Peleg and J. D. Ullman, An optimal synchronizer for the hypercube, Proc. of 6th Symp. on Principles of Distributed Computing (PODC), pp. 77–85, (1987).

[14] F. P. Preparata and M. I. Shamos, Computational Geometry, *Springer Verlag*, (1985).

[15] R. C. Prim, Shortest Connection Networks and Some Generalizations, *Bell System Tech. J.*, 36, pp. 1389–1401 (1957).