ABSTRACT

Robert Hochberg. Minimum Linear Arrangement of Trees. (Under the direction of Professor Matthias Stallmann.)

In the minimum linear arrangement problem one is given a graph, and wishes to assign distinct integers to the vertices of the graph so that the sum of the differences (in absolute value) across the edges of the graph is minimized. This problem is known to be NP-complete for the class of all graphs, but polynomial for special classes of graphs, one of which is the class of trees. For trees on n vertices, algorithms of time complexity $O(n^{2.2})$ and $O(n^{1.6})$ were given by Shiloach in 1979 and Chung in 1983 respectively, with no improvement since then. In this thesis, we present a linear-time algorithm for finding the optimal embedding among those embeddings which have no "crossings," and we describe a C++ implementation of that algorithm as well as Shiloach's algorithm which we make available to the research community.

MINIMUM LINEAR ARRANGEMENT OF TREES

by

Robert Hochberg

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh

2002

APPROVED BY:

Chair of Advisory Committee

To Valerie

BIOGRAPHY

Robert A. Hochberg

1985 Graduated from Sachem High School, Long Island, New York.

1985-88 Attended the State University of New York at Stony Brook, Stony Brook, New York.

1988 Attended the Technical University of Budapest.

1988 B.S., SUNY Stony Brook.

1988-94 Attended Rutgers University, the State University of New Jersey.

1994 Ph.D. in Mathematics, Rutgers University.

2001-02 Attended North Carolina State University

2002 M.S. in Computer Science, North Carolina State University

ACKNOWLEDGEMENTS

Thanks to Prof. Matthias F. M. Stallmann for taking the time to direct this thesis, for providing inspiration and helpful suggestions, and for having the quickest email turnaround time imaginable!

Thanks also to Professors Carla Savage and Dennis Bahler for agreeing to be on my thesis committee and for suggestions which led to a significantly improved (and correct) final document.

And thank you to Valerie, my kind and patient wife, for making many sacrifices to give me the time to work on this degree.

Contents

List of Figures

Chapter 1

Introduction

Given a graph $G = (V, E)$, a *linear arrangement* π of G is a bijection $\pi : V \to \{1, 2, \ldots, n\}$, where $n = |V|$. The *cost* of a linear arrangement π is given by the sum

$$C[\pi, G] = \Sigma_{(u,v)\in E}|\pi(u) - \pi(v)|,$$

and a *minimum linear arrangement* π of G is a linear arrangement which minimizes this sum.

For the class of all graphs, the problem of finding a minimum linear arrangement was shown in 1976 by Garey *et al.* [4] to be NP-complete. In that same year, Goldberg and Klipker [5] gave an $O(n^3)$ algorithm that solved the problem when G is a tree. Shiloach [8] improved this bound to $O(n^{2.2})$ in 1979, and Chung [2] further improved it in 1983 to $O(n^\lambda)$ for any λ satisfying $\lambda > \frac{\log 3}{\log 2} = 1.585\ldots$. There seems to have been no further reduction in the time complexity. Recently, Shahrokhi *et al.* [7] showed that an algorithm for minimum linear arrangement could be used to find the bipartite crossing number of trees, thus showing that an $O(n^\lambda)$ algorithm also exists for that problem.

The present work makes the following contributions to the study of this problem: In chapter 2, we discuss the problem of embedding trees "on one page," and give a linear time algorithm for this restricted problem. This is an improvement, for the case of trees, of a 1988 result of Frederickson and Hambrusch [3] which gives an optimal one-page embedding of any outperplanar graph in time $O(n^2)$. In chapter 3, we give an implementation of this one-page algorithm as well as an implementation of Shiloach's (optimal) embedding algorithm in the form of a library of functions ready for use by researchers. We implement Shiloach's algorithm rather than Chung's because it is conceptually simpler, and because

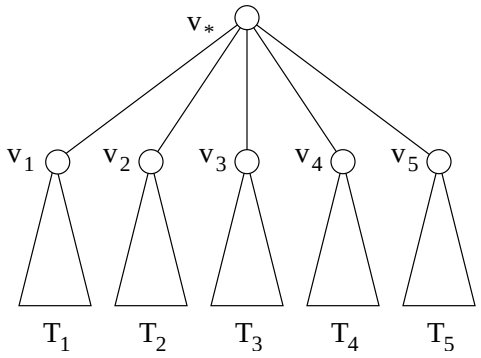


Figure 1.1: The subtrees of a tree mod a vertex

our purpose for implementing an optimal algorithm was simply for comparative analysis with our linear-time algorithm.

1.1 Notation

We follow the notation used by Shiloach in [8]. Let v_* be some vertex of a tree T, and consider the subtrees generated by deleting v_*. Each of them is called a *subtree of* T *mod* v_*, and for each of these subtrees T_i there is a vertex $v_i \in T_i$ such that the edge $(v_i, v_*) \in T$. The vertex v_i is called the *root of* T_i *mod* v_*. We denote by n_i the number of vertices in the tree T_i. See figure 1.1.

In what follows, we will tend to use the phrase "embedding" instead of "linear arrangement." Hence we will speak of optimal embeddings and one-page embeddings, rather than optimal or one-page linear arrangements. The terms are, for our purposes, synonymous.

Chapter 2

One-Page Embeddings

Let T be a tree and suppose we wish to find an embedding π such that all the edges of T can be drawn on one side of the number line without any pair of edges crossing. See figure 2.1. Such embeddings are called "one-page" embeddings and are a special case of "book embeddings" [1].

More rigorously, we can define π to be a one-page embedding if there do not exist four vertices $a, u, b, v \in T$ such that (a, b) and (u, v) are edges in T, and $\pi(a) < \pi(u) < \pi(b) < \pi(v)$. The following characterization is more useful for our purposes, and is easily shown to be equivalent to the definition given above.

Fact 1 *An embedding π is a one-page embedding if and only if there do not exist four vertices $a, u, b, v \in T$ with vertex-disjoint paths P_1 from a to b and P_2 from u to v, and $\pi(a) < \pi(u) < \pi(b) < \pi(v)$.*

Since this restricts the number of embeddings we are able to consider, we expect that the minimum cost over one-page embeddings could be greater than that over all embeddings. In fact, the tree shown in Figure 2.1 is such an example: The two embeddings shown are optimal with respect to allowing crossings and disallowing crossings, yet they differ in cost by 1.

On the other hand, we give in this chapter a linear time algorithm for finding an optimal, one-page embedding of a tree. This is in contrast to the best-known algorithm for general embeddings, which is of time complexity $O(n^{1.6})$. The author believes that a linear time algorithm for the general problem is possible.

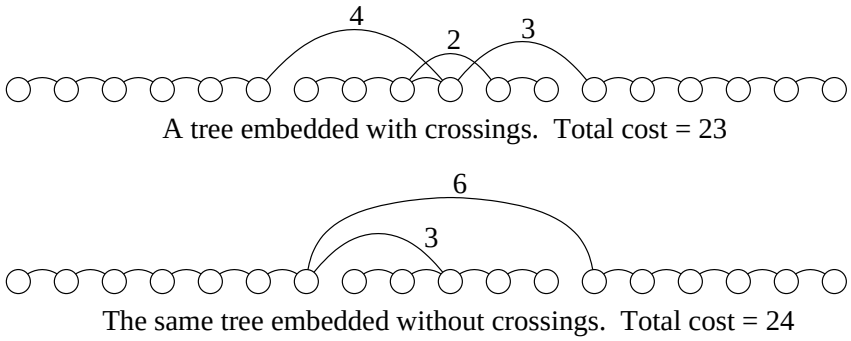


Figure 2.1: The same tree embedded with and without crossings

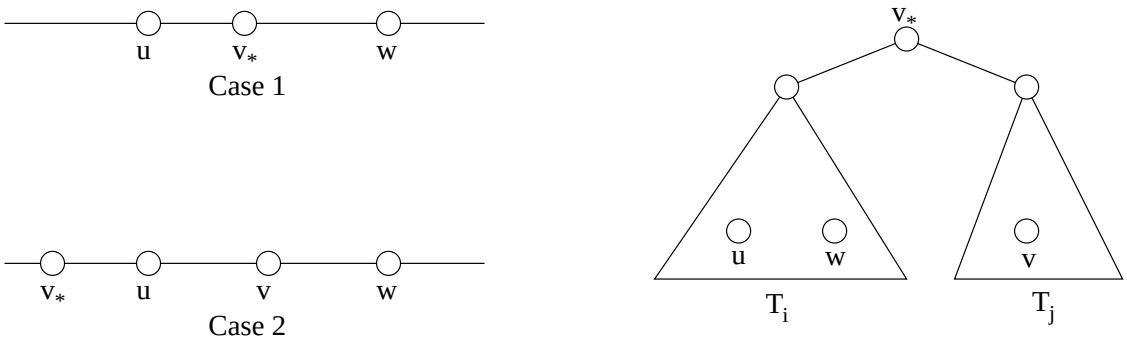


Figure 2.2: The proof of Lemma 2.

2.1 Some Observations and Theorems

Throughout this chapter we will assume that our tree has n vertices. Let T be a tree, let v_* be a vertex of T, and let $T_1, T_2, \ldots$ be the subtrees of T mod v_*. Suppose π is an optimal, one-page embedding of T. We call a vertex v *visible from the top* if there are no edges $(u, w) \in T$ such that $\pi(u) < \pi(v) < \pi(w)$.

Lemma 2 *If vertex v_* is visible from the top, then each subtree T_i mod v_* occupies a single interval of integers. That is, $\pi(T_i) = \{s, s+1, \ldots, t\}$ for some integers $s \leq t$.*

Proof. Suppose to the contrary that there exist three vertices u, v, w with $\pi(u) < \pi(v) < \pi(w)$ such that $u, w \in T_i$ but $v \notin T_i$. We distinguish two cases (see figure 2.2):

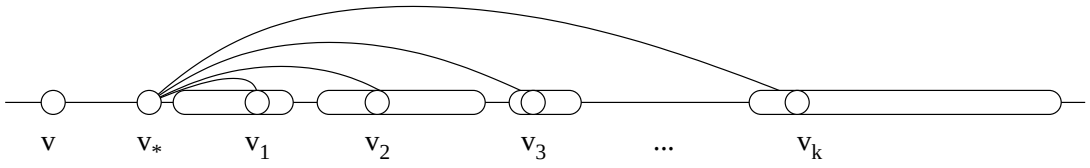


Figure 2.3: The proof of Lemma 3.

case 1 $\pi(u) < \pi(v_*) < \pi(w)$. Since u and w are in the same subtree mod v_*, there must be a path from u to w which does not include v_*. Let $u = x_1, x_2, \ldots, x_k = w$ be such a path, and let j be the least index such that $\pi(x_j) > \pi(v_*)$. Clearly $j > 1$. But then the edge (x_{j-1}, x_j) gives a contradiction to our assumption that v_* was visible from the top.
case 2 $\pi(v_*) < \pi(u)$ or $\pi(v_*) > \pi(w)$. We assume without loss of generality that $\pi(v_*) < \pi(u)$. Again, since u and w are in the same subtree mod v_*, there must be a path P_1 between them which lies entirely inside T_i. If v is in a different subtree, then there must be a path P_2 from v_* to v which is entirely disjoint from P_1. But then by Fact 1, we have a contradiction to π being a one-page embedding. □

We can say a little bit more about subtrees occupying intervals.

Lemma 3 *Let v_* be a vertex of T, not necessarily visible from the top, and suppose $T_1, T_2, \ldots, T_k$ are subtrees of T mod v_* with roots $v_1, v_2, \ldots, v_k$, such that $\pi(v_*) < \pi(v_1) < \pi(v_2) < \cdots < \pi(v_k)$. Then each of the subtrees $T_1, T_2, \ldots, T_{k-1}$ occupies a consecutive interval of integers. (See figure 2.3)*

Proof. The proof is almost the same as for Lemma 2. First note that the edge (v_*, v_k) precludes case 1 in that proof from occurring for any subtree other than T_k. That is, if some subtree T_i for $i \leq 1 \leq k-1$ contained a vertex v to the left of v_*, then by Fact 1, the quadruple of vertices v, v_*, v_i, v_k would yield a crossing in our embedding. Thus case 2 must obtain, and it is resolved here the same way it is resolved for Lemma 2.

Lemma 4 *The vertices $v = \pi^{-1}(1)$ and $w = \pi^{-1}(n)$ have degree 1 in T.*

Proof. Suppose not. Then, since v is visible from the top, we can find two subtrees T_1 and T_2 mod v with $\pi(T_1) = \{2, 3, \ldots, i\}$ and $\pi(T_2) = \{i+1, i+2, \ldots, j\}$. But then we can create a new embedding π' which has lower cost, contradicting the optimality of π. We do this by "flipping" the tree T_1 across to the other side of v. (see figure 2.4) Indeed, define

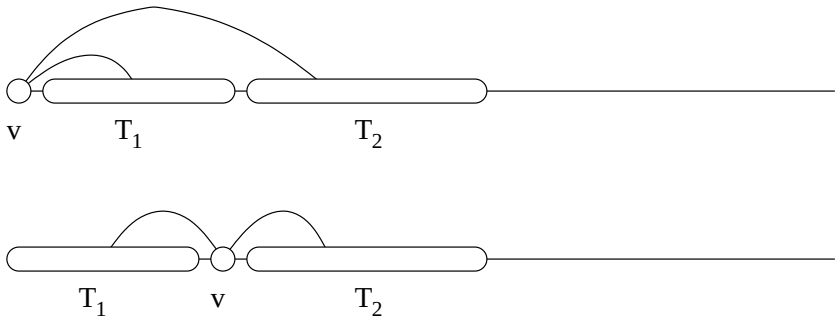


Figure 2.4: Proof of Lemma 4. We can flip subtree T_1 to the other side of v to obtain an embedding with lower total cost

$$\pi'(x) = \begin{cases} i & \text{if } x = v \\ i + 1 - \pi(x) & \text{if } x \in T_1 \\ \pi(x) & \text{otherwise} \end{cases}$$

All the lengths within each subtree remain the same, as do all the distances from v to the roots of the subtrees, except for the distance from v to the root of T_2. This length is decreased by $i - 1$. This completes the proof for v. The proof for w is the same. □

The path between v and w in the Lemma above is called the *basic path* of the embedding, and is guaranteed to be monotone since the embedding has no crossings. The vertices on this basic path are exactly those vertices which are visible from the top. Lemma 4 is just a special case of a more general theorem:

Lemma 5 *If v is a vertex of T, then in any optimal embedding π, the number of subtrees mod v which lie to the left of v is within 1 of the number of subtrees which lie to the right of v. (We note that this implies equality if v is of even degree.)*

Proof. Suppose not. Then we use the same idea as we did in the proof of Lemma 4. Suppose that there are more subtrees to the right of v than to the left (in which case there must be at least two subtrees to the right) and that T_1 is that subtree on the right of v which is closest to v, that is, contains the integer $\pi(v) + 1$. (See figure 2.5) By Lemma 3, the vertices of T_1 occupy a single interval of integers. If we flip T_1 over to the other side of v, then the only edges which change length are those which connect v to the root of a subtree mod v other than T_1. Those edges going to a subtree on the right will decrease

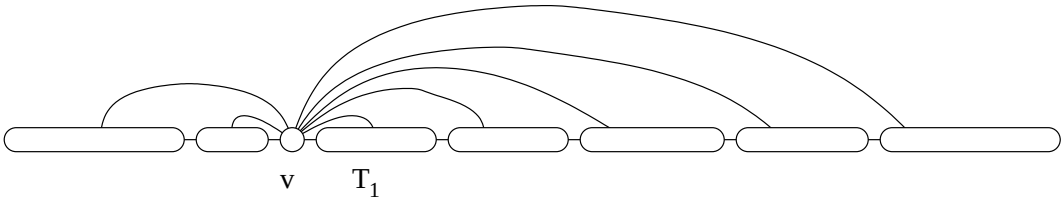


Figure 2.5: Proof of Lemma 5. In an optimal embedding, vertices should have the same number of subtrees on each side

by $|V(T_1)|$, while those edges going to the left will increase by the same length. But by assumption, the number that decrease in length is greater than the number that increase in length, contradicting the optimality of π. □

Given a vertex v which is visible from the top, we know that its subtrees must occupy consecutive intervals of integers, and that there must be the same number of them (within 1) embedded on each side of v. We now determine the order in which these subtrees must be embedded in an optimal embedding.

Lemma 6 *Let π be an optimal one-page embedding of a tree T, and let v_* be a vertex on the basic path. Let $T_1, T_2, \ldots, T_k$, be the subtrees of T mod v_* with $n_1 \geq n_2 \geq \cdots \geq n_k$. Then we may assume that the vertex v_* and its subtrees are embedded in the order*

$$\begin{aligned} &(T_1, T_3, \ldots, T_k, v_*, T_{k-1}, \ldots, T_4, T_2) \quad \textit{if } k \textit{ is odd} \\ &(T_1, T_3, \ldots, T_{k-1}, v_*, T_k, \ldots, T_4, T_2) \quad \textit{if } k \textit{ is even;} \end{aligned}$$

that is, there is an embedding of T in the form given which is an optimal embedding.

Proof. We prove the claim by modifying the embedding π in several steps to obtain an embedding of the form desired. We also show that each step does not increase the cost of the embedding. We distinguish steps of two types:

Type A Suppose $1 \leq i < j \leq k$, and that T_i and T_j are two subtrees which are adjacent to one another in the embedding, with T_i closer than T_j to v_*. (See figure 2.6a.) Then we may swap the order of those two subtrees, keeping all other vertices in the same relative positions. The only edges that change lengths under this step are (v_*, v_i) which increases in length by n_j, and (v_*, v_j) which decreases in length by n_i. But since $i < j$, we have $n_i \geq n_j$, and we have not increased the cost of the embedding. In fact, we will have decreased the

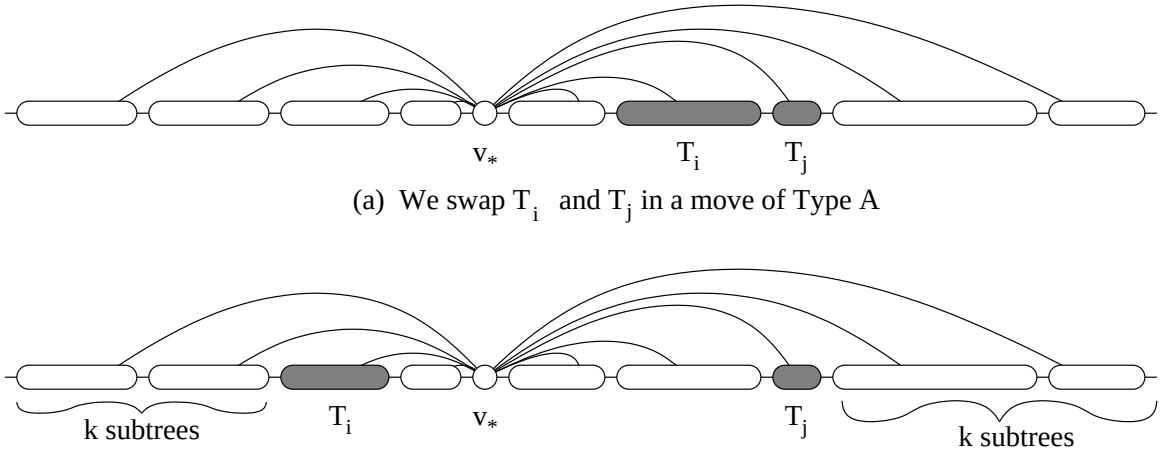


Figure 2.6: Proof of Lemma 6. Two ways to rearrange the order of the subtrees without increasing the total cost

cost of the embedding if n_i is strictly less than n_j. (This last fact is used in the proof of Lemma 7 below.)

Type B Suppose T_i and T_j are two subtrees which lie on the left and right sides of v_* respectively, such that the number of subtrees that lie to the left of T_i is the same as the number of subtrees that lie to the right of T_j. Suppose there are k such subtrees on each side. (See figure 2.6b.) Then we may switch and flip these two subtrees, keeping all other vertices in the same relative positions. The sum of the costs of edges (v_*, v_i) and (v_*, v_j) will remain unchanged, since the number of vertices between the two subtrees did not change. The only other edges which will change lengths are the edges $e_l = (v_*, v_l)$ as v_l ranges over the roots of the trees to the left of T_i and to the right of T_j. If v_l lies to the left of T_i, then the edge e_l will change length by $(n_j - n_i)$. If v_l lies to the right of T_j, the edge e_l will change length by $(n_i - n_j)$. These changes exactly cancel out.

We now prove the theorem as follows. First apply steps of type A on each side of v_* to move the subtrees of greatest size, including T_1, to the "outside," furthest away from v_*. Then apply a step of type B, if necessary, so that T_1 ends up in its proper place. To get T_2 to its proper position, first make sure that it is on the right side of v_*, by applying a step of type B, if necessary. Note that this will not affect the position of T_1. Then we apply any necessary steps of type A so that T_2 moves to the outside position on the right side. Again,

we will not affect the position of T_1. We note that if there are several trees of sizes n_1 or n_2 then we may be doing some steps that switch subtrees of the same size, just because of the way we indexed our trees. Proceeding in this fashion, we can sequentially move each T_i to its predicted position without increasing the cost of the embedding, yielding another optimal embedding of the predicted type. □

We next prove a partial converse of Lemma 6 which we need in order to prove that the central vertex, discussed in the next section, lies on the basic path.

Lemma 7 *Let v_* be a vertex of degreee $d > 1$ on the basic path of an optimal one-page embedding, and let*

$$U_1, U_2, \ldots, U_s, v_*, U_{s+1}, \ldots, U_d$$

be the order in which its subtrees are embedded. Then for each $i, 1 < i < d$,

$$|U_i| \leq |U_1| \text{ and } |U_i| \leq |U_d|.$$

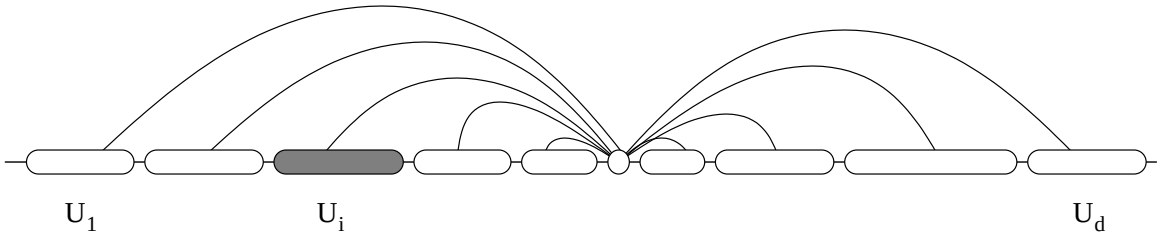
Proof. Suppose not. Then there is some subtree U_i such that either $|U_i| > |U_1|$ or $|U_i| > |U_d|$. Let us suppose without loss of generality that $|U_i| > |U_1|$. Then there are two cases, as shown in Figure 2.7:

case 1 U_i lies to the left of v_*. Then we swap the two subtrees, leaving all other subtrees in their same relative positions. Let $L = |U_2| + |U_3| + \cdots + |U_{i-1}|$ and denote by v_j the root of subtree U_j mod v_*. Then (v_*, v_1) will decrease in length by $L + |U_i|$ and (v_*, v_i) will increase in length by $L + |U_1|$, for a net decrease in cost. Additionally, the edges (v_*, v_j) for $2 \leq j \leq i-1$ will decrease in length by $|U_i| - |U_1|$. This overall decrease in cost contradicts the optimality of the original embedding.

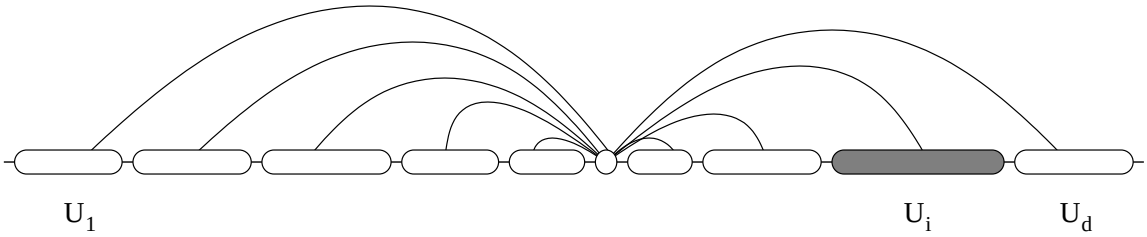
case 2 U_i lies to the right of v_*. Then we swap and flip the two subtrees, and in the same fashion as in Case 1, the total overall cost would decrease, again contradicting optimality.□

2.2 The Central Vertex

A *central vertex* c of an n-vertex tree T is a vertex such that all of the subtrees of T mod c have size at most $n/2$. We note that this is different from what is commonly called a *center* of a tree, which is a vertex v that minimizes $\max_{u \in T} d(v, u)$. It is a well-known theorem that every tree has exactly one or two centers. It turns out that every tree also has exactly one or two central vertices, though they need not be the same as the center(s). We



Case 1: We swap subtrees U_1 and U_i to obtain an embedding of lower total cost



Case 2: We swap and flip subtrees U_1 and U_i to obtain an embedding of lower total cost

Figure 2.7: The proof of Lemma 7

follow Shiloach [8] in calling them "central vertices." Shiloach proves that every tree has a central vertex. We give our own proof below, because it anticipates some of the ideas used in the next theorem. We will also prove that a tree has no more than two centers. First, let us introduce some new notation.

For every pair of vertices u, v for which (u, v) is an edge of T, define $s(u, v)$ to be the number of vertices in the subtree of T mod u which contains v. For example, in Figure 2.8, $s(u, v) = 3$, and $s(v, u) = 7$. We note the following:

$$\text{for any edge}(u, v) \in T, \quad s(u, v) + s(v, u) = \quad n \tag{2.1}$$

$$\text{for any vertex } u \in T, \quad \textstyle\sum_{(u,v)\in T} s(u, v) = \quad n - 1 \tag{2.2}$$

Lemma 8 *Every tree has a central vertex.*

Proof. Select any vertex v_1 of T. If v_1 is a central vertex, we are done. Otherwise, there is some edge (v_1, v_2) (in fact, exactly one) for which $s(v_1, v_2) > n/2$. Then $s(v_2, v_1) =$

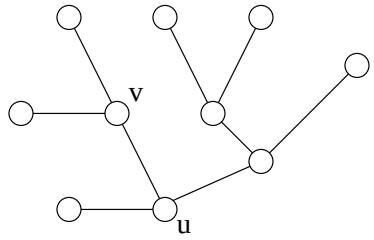


Figure 2.8: An example of subtree sizes: $s(u, v) = 3$ and $s(v, u) = 7$

$n - s(v_1, v_2) < n/2$. Is v_2 a central vertex? If so, we are done. If not, then we can find some edge (v_2, v_3) for which $s(v_2, v_3) > n/2$ and $s(v_3, v_2) < n/2$. Is v_3 a central vertex? If so, we are done. If not, then let us continue in this fashion, generating a sequence $v_1, v_2, v_3, \ldots$ of vertices. This sequence forms a path in the tree, since $v_i \neq v_{i+1}$ and the tree is acyclic. Clearly it cannot continue until it reaches a leaf, because then $s(v_i, v_{i+1}) = 1 < n/2$. So the process must terminate at some vertex v for which no edge (v, w) has $s(v, w) > n/2$, and v is a central vertex of our tree. □

Lemma 9 *No tree has more than two central vertices.*

Proof. Let c be a central vertex of the tree, and consider any path $c = x_1, x_2, \ldots$ starting from c. Suppose $s(c, v) < n/2$ (strict inequality) for all v adjacent to c. Then we have $s(x_1, x_2) < n/2$ which implies $s(x_2, x_1) > n/2$ by (2.1), and so x_2 is not a center. But then $s(x_2, x_3) < n/2$ by (2.2), which implies $s(x_3, x_2) > n/2$, and so x_3 is not a center, and so on. This proves that no other vertex in T can be a center. But what if there is some vertex v such that $s(c, v) = n/2$ (there can be at most one by 2.2). Then $s(v, c) = n/2$, and for all other edges $(v, w), s(v, w) < n/2$, again by 2.2. Thus v is another center, but the only other center. □

We now come to the theorem which shows why centers are relevant to the present problem.

Theorem 10 *In any optimal embedding π of T, all centers of T must lie on the basic path.*

Proof. Let $x_1, x_2, \ldots, x_k$ be the basic path. By Lemma 4, $s(x_1, x_2) = n - 1$ and $s(x_{k-1}, x_k) = 1$. Thus we can define i as the least index such that $s(x_i, x_{i+1}) \leq n/2$. But then $s(x_{i-1}, x_i) > n/2$ which implies $s(x_i, x_{i-1}) < n/2$. We can show that x_i is a center

by showing that all other edges (x_i, v) out of x_i satisfy $s(x_i, v) < n/2$. But this follows directly from Lemma 7 and the inequalities just mentioned.

What if T has two centers? Then x_i will have one edge (x_i, v) such that $s(x_i, v) = n/2$. By Lemma 7, this will be edge (x_i, x_{i+1}), with $s(x_{i+1}, x_i) = n/2$, making x_{i+1} the other center, also on the basic path. □

2.3 Anchored Trees

Our algorithm for finding an optimal, one-page embedding begins by embedding a central vertex c of the graph, and then embedding its subtrees according to Lemmas 5 and 6. A small bit of care must be taken, however, when embedding these subtrees. This is because the edge between c and the root of a subtree mod c has a length that depends on the embedding of that subtree, and this length contributes to the overall cost of the embedding. We are thus led to consider embeddings of *left-anchored trees* and *right-anchored trees*, defined as follows: A left-anchored tree is a tree with a distinguished vertex v_* and an extra edge that joins v_* to a vertex which liesto the left of the embedding. Right-anchored trees are defined symmetrically. The next two theorems describe optimal embeddings of left-anchored trees. They are analogous to Lemmas 5 and 6 for non-anchored trees. The theorems for right-anchored trees are symmetric.

Lemma 11 *If π is a left-anchored embedding of T with anchor v_*, then the number of subtrees that lie to the right of v_* is either the same as or one more than the number of subtreees that lie to the left of v_*.*

Proof. The proof is essentially the same as the proof of Lemma 5, taking into consideration the "anchor edge," which also contributes to the cost of the embedding.

Lemma 12 *Let π be an optimal one-page embedding of a left-anchored tree T with anchor v_*. Let $T_1, T_2, \ldots, T_k$, be the subtrees of T mod v_* with $n_1 \geq n_2 \geq \cdots \geq n_k$. Then we may assume that the vertex v_* and it subtrees are embedded in the order*

$$\begin{aligned} &(T_2, T_4, \ldots, T_{k-1}, v_*, T_k, \ldots, T_3, T_1) \quad \textit{if } k \textit{ is odd} \\ &(T_2, T_4, \ldots, T_k, v_*, T_{k-1}, \ldots, T_3, T_1) \quad \textit{if } k \textit{ is even;} \end{aligned}$$

that is, there is an embedding of T in the form given which is an optimal embedding.

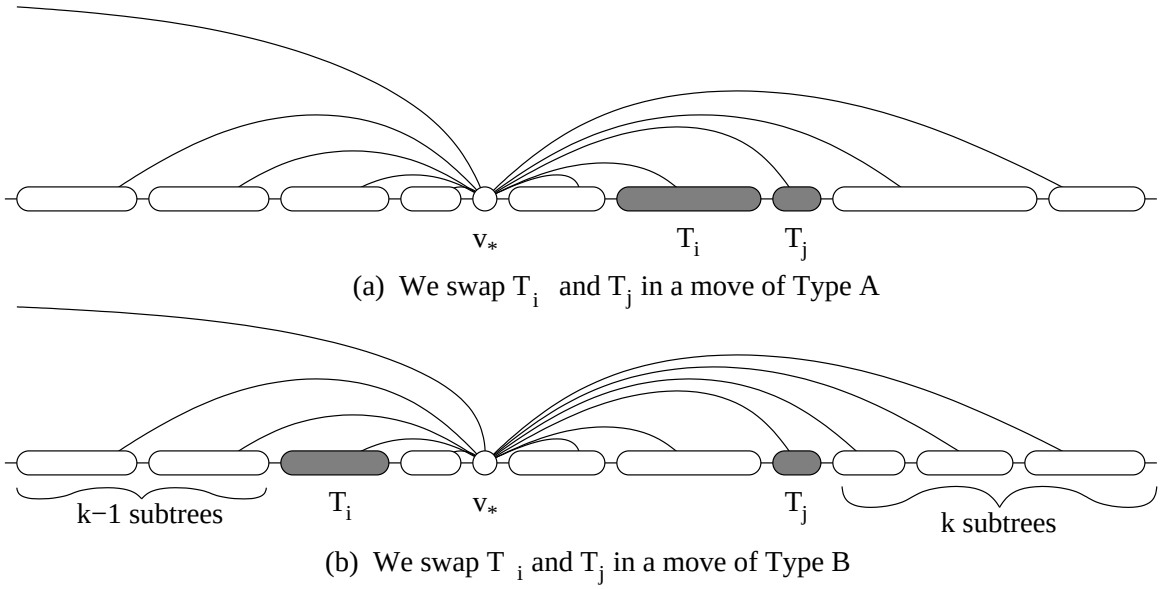


Figure 2.9: Proof of Lemma 12. Two ways to rearrange the order of the subtrees without increasing the total cost

Proof. We prove the claim by modifying the embedding π in several steps to obtain an embedding of the form desired, just as in the proof of Lemma 6. Steps of type A are the same, and steps of type B are only slightly different. See Figure 2.9.

Type A If T_i and T_j are two subtrees which are embedded adjacent to one another, with T_i closer than T_j to v_*, but $n_j \geq n_i$ then we may swap the order of those two subtrees, keeping all other vertices in the same relative positions. Cost will not increase, as shown in the proof of Lemma 6.

Type B Suppose T_i and T_j are two subtrees which lie on the left and right sides of v_* respectively, such that the number of subtrees that lie to the left of T_i is one less than the number of subtrees that lie to the right of T_j. (See figure 2.9b.) Then we may switch and flip these two subtrees, keeping all other vertices in the same relative positions. The proof that the cost will not increase is the same as in Lemma 6, with the anchor edge playing the role of the k'th subtree edge. The rest of the proof proceeds exactly as in the proof of Lemma 6. □

The following theorem summarizes all that has been said in the preceeding lemmas, and completely characterizes optimal one-page embeddings of trees.

Theorem 13 (Main Theorem) *Let π be an optimal embedding of a tree T with central vertex c and subtrees $T_1, \ldots, T_k$ mod c, with $n_1 \geq n_2 \geq \cdots \geq n_k$ as usual. Then we may make the following assumptions about π:*

1. *c must be on the basic path*
2. *The subtrees mod c lie as described in Lemma 6*
3. *Each subtree mod c is embedded as described in Lemma 12*

Proof All that needs special proof is part 3. To that end we note that the embedding of each subtree mod c is in fact an anchored embedding. If any of these are not embedded as described in Lemma 12, then we can modify that subtree's embedding to obtain an embedding of the desired type without increasing the cost. □

2.4 The Algorithm

We now give an algorithm for finding an optimal one-page embedding of a tree T in time linear in n, the number of vertices of T. Here is the big picture; we give the implementation details below.

1. Locate a central vertex
2. Embed that central vertex
3. Embed each vertex adjacent to the central vertex
4. Recursively embed each subtree mod the central vertex

2.4.1 Locate a central vertex

To locate a central vertex, we first compute $s(u, v)$ and $s(v, u)$ for all edges (u, v) of T. We do this by selecting an arbitrary vertex of T, and doing a depth-first-search from that vertex. During the DFS, whenever u is the parent of v, we use the relation

$$s(u, v) = 1 + \sum_{\substack{\text{edges } (v, w) \\ w \neq u}} s(v, w).$$

to find $s(u, v)$, and then use

$$s(v, u) = n - s(u, v)$$

to find $s(v, u)$.

get_subtree_sizes

procedure get_subtree_sizes
    Set n = number of vertices of T
    Set $s(u, v) = 0$ for all edges (u, v)
    $root \leftarrow$ some vertex of T
    for each vertex w adjacent to $root$
        get_subtree_size_dfs($root, w$)
end

function get_subtree_sizes_dfs($parent, v$)
    $returnvalue = 0$
    for each vertex w adjacent to v (other than $parent$)
        $returnvalue$ += get_subtree_sizes_dfs(v, w)
    $s(parent, v) = returnvalue + 1$
    $s(v, parent) = n - s(parent, v)$
    return $returnvalue + 1$
end

To find the central vertex, we use the algorithm given in the proof of Lemma 8, which takes time $O(n)$. The time complexity of get_subtree_sizes is the same as for a DFS, which is $O(n)$.

2.4.2 Embed the central vertex

To get the placement of the central vertex, c, we use Lemma 6, and simply take $\pi(c) = n_1 + n_3 + \cdots + n_l + 1$, where l is either k or $k - 1$, whichever is odd.

Note that before computing this quantity, we perform a bucket sort on the edge lists at each vertex according to subtree size. That this can be done for the entire graph in linear time was shown by Yao in 1975 [9]. Thus the loop that computes the sum above can be done in time proportional to the degree of c.

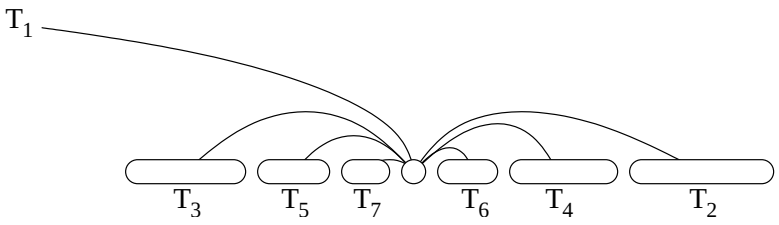


Figure 2.10: The "before" information is the sum of the sizes of the subtrees under the anchor edge

2.4.3 Embed each vertex adjacent to the central vertex

In order to embed these vertices, we need to calculate the *before* value $b(v)$ of each vertex v of T. If $T_1, T_2, \ldots, T_k$ are the subtrees of T mod v, with $n_1 \geq n_2 \geq \cdots \geq n_k$, then

$$b(v) = \sum_{\substack{i \text{ odd} \\ 3 \leq i \leq k}} n_i.$$

Indeed, $b(v)$ gives the number of vertices that come before the anchor in an embedding of the subtree, and is the additional number of nodes that the anchor edge must cross. To see that its value is as given by the equation above, note that we begin our embeddings from a central vertex of the tree, and therefore all anchor edges will point back toward the central vertex. So for any node (other than the central vertex) the anchor edge will go to T_1, that is, the largest subtree of that node, and by Lemma 12, the rest of the subtrees will embed in the order shown.

To find where each of the vertices adjacent to the central vertex is embedded we use a single loop, as shown below. The quantities $leftsum$ and $rightsum$ keep track of the numbers of vertices so far embedded on each side of the central vertex, for use when embedding the next subtree.

Embed the vertices adjacent to the central vertex

procedure embed_one_page(T)
    $leftsum = rightsum = 0$
    for $i = k$ **downto** 1
        if (i is even)
            $\pi(v_i) = \pi(c) + rightsum + b(v_i) + 1$;
            $rightsum = rightsum + n_i$
            embed_one_page_subtree(T_i, v_i, right)
        if (i is odd)
            $\pi(v_i) = \pi(c) - leftsum - b(v_i) - 1$;
            $leftsum = leftsum + n_i$
            embed_one_page_subtree(T_i, v_i, left)
end

This loop runs in time $O(k)$, not including the recursive calls in each "if" block. The parameters"right" and "left" indicate to the subtree procedure the direction in which the anchor edge goes. We will see in the next section how this parameter is used.

2.4.4 Recursively embed each subtree mod the central vertex

The recursive step uses a loop much like that given above. The difference is that the loop goes from k to 2, because we don't embed the "anchor tree," T_1. We must also keep track of whether our anchors are going to the right or to the left. The quantities $rightsum$ and $leftsum$ play the same role as in the above algorithm, and the parameter "dir" takes on the values "left" and "right," indicating the direction of the anchor edges.

Embedding subtrees

procedure embed_one_page_subtree(T, v, dir)
▷ Suppose T has subtrees $T_1, \ldots, T_k$ with $n_1 \geq \cdots \geq n_k$.
$leftsum = rightsum = 0$
for $i = k$ **downto** 2
if ((i is even **and** dir = right) **or** (i is odd **and** dir = left))
$\pi(v_i) = \pi(v) + rightsum + b(v_i) + 1$;
$rightsum = rightsum + n_i$
embed_one_page_subtree(T_i, v_i, right)
if ((i is even **and** dir = left) **or** (i is odd **and** dir = right))
$\pi(v_i) = \pi(v) - leftsum - b(v_i) - 1$;
$leftsum = leftsum + n_i$
embed_one_page_subtree(T_i, v_i, left)
end

The time spent at each node (not including the time spent at the descendents of that node and not including the overhead of doing a DFS) is proportional to the number of children of that node. So again we can use aggregate analysis to conclude that the total time complexity, in addition to the "overhead" time complexity of using a DFS, is equal to the total number of children of all nodes, which is equal to the number of edges, which is $O(n)$. Performing a DFS on a tree also takes time $O(n)$, so our total time is linear in n. Since every step of our algorithm had a linear time bound, we conclude that our algorithm as a whole is linear.

Chapter 3

Implementation of Optimal Embedding Algorithms

This chapter briefly describes our implementation of three algorithms that we have made available as a C++ library. They are get_center(), embed_one_page() and embed().

3.1 Environment

These functions were written in C++ using the *Library of Efficient Data Types and Algorithms* (LEDA) [6]. The work described here is implemented using Release 4.1, one of the latest available free releases.[1] For our purposes, LEDA provides built-in datatypes and algorithms for graphs, arrays indexed by the nodes or edges of graphs, sort routines and lists. Our library functions take LEDA graphs as arguments, so the user must have LEDA installed on his machine.

3.2 Functions

We now describe three functions that are available to the user, basically in the form of additional LEDA library routines.

[1] On the NCSU EOS/Unity system, do `add leda`.

3.2.1 The "get_center" function

```
node get_center(graph &G)
```

This function works as described in Section 2.2. It first finds all subtree sizes of T, then chases along some path to find a central vertex, c. The central vertex is returned. The running time is linear in n, the number of vertices of T.

get_center

function get_center(T)
    get_subtree_sizes(T)
    $v \leftarrow$ any vertex of T
    while (v is not a center)
        Find $e = (v, w)$ such that $s(v, w) > n/2$
        $v \leftarrow w$
    end
    return v
end

3.2.2 The "embed_one_page" function

```
int embed_one_page(graph &G, node_array<int> &placement)
```

This function implements the algorithm described in Section 2.4. All the pseudocode is given there. Its first argument is a graph, which should be a tree. Its second argument is a node_array which the algorithm will fill with the embedding. So, if v is some node of T, then placement[v] will equal $\pi(v)$ for some optimal embedding π. The function runs in linear time, and returns the total cost of the embedding.

3.2.3 The "embed" function

```
int embed(graph &G, node_array<int> &placement)
```

This function works the same way (from the user's point of view) as embed_one_page(),

except that it uses Shiloach's algorithm to return an optimal embedding with or without crossings. It also returns the cost of the embedding, but does not run in linear time. We first give a brief description of the ideas behind Shiloach's algorithm. He calls trees which are not anchored "free" trees. In the second theorem, we do not include the subtree attached to the anchor in the list $T_1, T_2, \ldots$ of subtrees mod v_*.

Shiloach proved the following

Theorem 14 (Free trees) *Suppose T is a free tree with central vertex c and subtrees $T_1, T_2, \ldots, T_k$ mod c, with $n_1 \geq n_2 \geq \ldots \geq n_k$ as usual. Let p be the greatest integer satisfying*

$$n_{2p+1} > \left\lfloor \frac{n_1 + 2}{2} \right\rfloor + \left\lfloor \frac{n_* + 2}{2} \right\rfloor \text{ where } n_* = n - \sum_{i=1}^{2p+1} n_i$$

If no such p exists, then set $p = 0$. Then T has an optimal embedding in which one of the following cases holds:

A: *$\pi(T_1) = \{1, 2, \ldots, n_1\}$ (Figure 3.1(a)), or*

B: *subtrees T_2 through T_{2p+1} are embedded as shown in Figure 3.1(b). Note that each subtree occupies a consecutive interval of integers, and here are no vertices outside the indicated subtrees.*

If $p = 0$, then the embedding must be of type A.

Theorem 15 (Anchored trees) *Suppose T is anchored from the right with anchor v_* and subtrees $T_1, T_2, \ldots, T_k$ mod c, with $n_1 \geq n_2 \geq \ldots \geq n_k$ as usual. Let p be the greatest integer satisfying*

$$n_{2p} > \left\lfloor \frac{n_1 + 2}{2} \right\rfloor + \left\lfloor \frac{n_* + 2}{2} \right\rfloor \text{ where } n_* = n - \sum_{i=1}^{2p} n_i$$

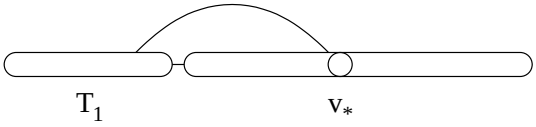
If no such p exists, then set $p = 0$. Then T has an optimal embedding in which one of the following cases holds:

A: *$\pi(T_1) = \{1, 2, \ldots, n_1\}$ (Figure 3.1(c)), or*

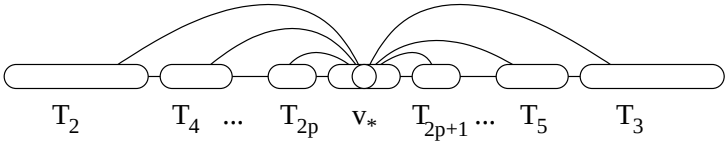
B: *subtrees T_2 through T_{2p} are embedded as shown in figure 3.1(d). Note that each subtree occupies a consecutive interval of integers, and there are no vertices outside the indicated subtrees.*

If $p = 0$, then the embedding must be of type A.

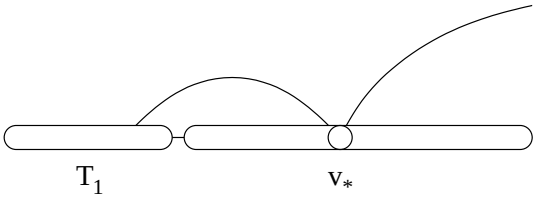
The pseudocode given below mirrors the algorithm given in Shiloach's paper [8]. We give here only the algorithm for free trees. The algorithm for anchored trees requires some minor modifications, as indicated by Theorem 15.



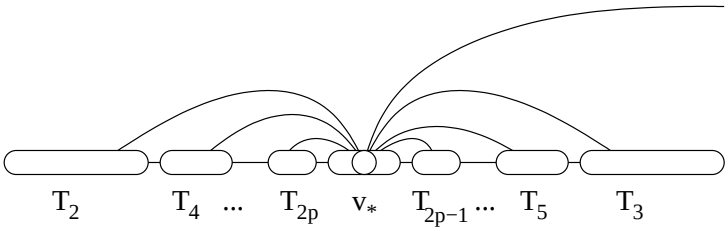
(a) Embedding a free tree, Case A.



(b) Embedding a free tree, Case B



(c) Embedding an anchored tree, Case A.



(d) Embedding an anchored tree, Case B.

Figure 3.1: Shiloach's Theorems. Parts (a) and (b) apply to Theorem 14, parts (c) and (d) apply to Theorem 15.

Shiloach's embedding algorithm for free trees

Find a central vertex c

Find optimal anchored embeddings for T_1 and $T - T_1$.

Let $costA$ be the cost of a Type A embedding

Find the value of p

if $p = 0$

    The arrangement is of Type A. Stop.

for i $= 2$ **to** $2p + 1$

    Find an optimal embedding of T_i

Find an optimal embedding of $T - (T_2, \ldots, T_{2p+1})$

Let $costB$ be the cost of a Type B embedding

if $costA < costB$

    The arrangement is of Type A. Stop.

else

    The arrangement is of Type B. Stop.

Bibliography

[1] Bernhart, F. and Kainen, P. C., The book thickness of a graph. J. Combin. Theory Ser. B 27 (1979), no. 3, 320–331.

[2] Chung, F. R. K., On optimal linear arrangements of trees. Comput. Math. Appl. 10 (1984), no. 1, 43–60.

[3] Frederickson, G. N. and Hambrusch, S. E., Planar linear arrangements of outerplanar graphs. IEEE Trans. Circuits and Systems 35 (1988), no. 3, 323–333.

[4] Garey, M. R., Johnson, D. S. and Stockmeyer, L., Some simplified NP-complete problems. Sixth Annual ACM Symposium on Theory of Computing (Seattle, Wash., 1974), pp. 47–63. Assoc. Comput. Mach., New York, 1974.

[5] Goldberg, M. K.; Klipker, I. A., An algorithm for minimal numeration of tree vertices. (Russian) Sakharth. SSR Mecn. Akad. Moambe 81 (1976), no. 3, 553–556.

[6] Mehlhorn, K. and Näher, S. LEDA: A platform for Combinatorial and Geometric Computing. Cambridge University Press, Cambridge, United Kingdom. 1999.

[7] Shahrokhi, Farhad; Sýkora, Ondrej; Székely, László A.; Vrťo, Imrich. On bipartite drawings and the linear arrangement problem. SIAM J. Comput. 30 (2001), no. 6, 1773–1789 (electronic).

[8] Shiloach, Yossi, A minimum linear arrangement algorithm for undirected trees. SIAM J. Comput. 8 (1979), no. 1, 15–32.

[9] Yao, Andrew C. An $O(E \log \log V)$ algorithm for finding minimum spanning trees. Information Processing Letters, 4, pp.21–23, 1975.