On Minimum Spanning Trees

D. Pe'er A. Wigderson

December, 1998

Abstract

The minimal spanning tree problem is one of the oldest and most basic graph problems in theoretical computer science. Its history dates back to Boruvka's algorithm in 1926 and till today it is a extensively researched problem with new breakthroughs only recently. Given an edge-weighted graph, this problem calls for finding a subtree spanning all the vertices, whose total weight is minimal. The central open question is: Does there exist a linear time deterministic algorithm that finds the minimal spanning tree? This problem remains open. We will present the problem from different angles giving our insight on each. We present the problem in a new framework, and improve the running time for specific families of graphs in this framework.

Contents

1 Overview

Given an edge-weighted graph, the Minimum Spanning Tree (MST) problem calls for finding a subtree spanning all the vertices, whose total weight is minimal. It is conjectured by many computer scientists that this problem has deterministic linear time complexity. We, as many others before us, set out to find such a linear time algorithm, using recently developed tools. In this paper we bring our observations and results from some of the novel viewpoints of MST which we have examined. Our hope is that this paper gives a better understanding of MST and may provide leads or intuition for future research.

We now give an overview of the sections in this thesis.

- In section 2 we introduce the MST problem. We define the problem and describe two basic combinatorial properties, the Cut Property and the Cycle Property, which allow us to classify edges as belonging to the MST or not. Finally, we present a survey of the existing deterministic algorithms from the first algorithm in 1926 [2] to the most recent one from 1997 [3]. In addition we present an important application of the cycle property, which identifies non-MST edges. We sketch an existing algorithm that uses this property to verify a given MST in deterministic linear time [13].

- In section 3 we show a linear time reduction from the MST problem to its restriction for 3-regular graphs. This reduction shows that very sparse graphs suffice to capture all the hardness of the MST problem.

- In section 4 we describe an existing randomized MST algorithmthat runs in expected linear time [12]. This algorithm is based on edge sampling and uses both the cycle and the cut properties. We present our proof of the main probabilistic lemma in this algorithm. Our proof is much simpler then [12], and provides deeper insight into the role of independence in this lemma.

- In section 5 we suggest a new framework for the MST problem. All known deterministic algorithms use a non-linear number of edge comparisons. We suggest studying the MST in the decision tree computational model. We formalize the MST problem in the language of posets by presenting a correspondence between spanning trees and posets. The Poset corresponding to a spanning tree is such that all its linear extensions are exactly the linear orders for which this tree is minimal. We define these posets and describe some of their basic characteristics.

- In section 6 we present two divide and conquer algorithms for graphs with small separators that find a MST using only $O(m)$ comparisons. These algorithms give a constructive linear depth decision tree for graphs of this family. Graphs with small separators constitute a large family of graphs for which all existing deterministic algorithms require a non-linear number of comparisons. An edge separator is a set of edges in the graph that after their removal the graph becomes disconnected. We use separators of sub-linear size that separate the graph into two components of approximately equal size. The general strategy is to remove the separator, solve the MST recursively for each connected component and efficiently patch the separator edges using the cycle property. The two algorithms differ in the way separator edges are handled and are linear time in slightly different conditions. For any $\epsilon > 1$, one algorithm runs in linear time for any G which is $\frac{m}{(\log m)^{2+\epsilon}}$-recursably separable, the other runs in linear time for any G which is $\frac{m}{(\log m)^{1+\epsilon}}$-strongly separable.

2 Introduction to MST

2.1 Problem Definition

We begin with some definitions and notations. All the graphs in this paper are finite, simple, and undirected. We denote by n the number of vertices, m the number of edges in a graph $G = (V, E)$. Denote the degree of the vertex v by $d(v)$. When a graph's vertices have a constant degree d we call it d-regular. Our graphs are weighted, $w(e)$ denoting the distinct weight of the edge e. The edge weights define a linear ordering on the edges. We identify a tree $T = (V, E)$ with its edges E. For $U \subseteq V$, denote the *cut* $(U, V \setminus U) = \{(u, v) \in E | u \in U, v \in V\}$.
MST Problem: *The* Minimal Spanning Tree *problem is:*
Input: *A weighted graph* (G, w).
Output: *The Unique spanning tree* T *that minimizes* $\sum_{e \in T} w(e)$.

When G is not connected we find the minimal spanning forest MSF: A set of trees, one in each of the connected component of G, each tree being a minimal spanning tree of the graph induced by the component.

2.2 Basic Properties

Given a graph G and any spanning tree T of G, we define the following sets:

- For any $e \notin T$, denote $cyc_T(e)$ as the set of edges in the unique cycle formed in $T \cup e$. When the spanning tree is clear from the context we omit the subscript. We use the same definition for forests.
- For any $e \in T$, whose removal from T creates the components U, $V \setminus U$, denote the cut $(U, V \setminus U)$ in G by $cut_T(e)$.

Definition 1 *If F is a forest in G, for an edge $e = (u, v)$ we denote by $p_F(e)$ the path (if any) connecting v and u in F, and by $w_F(e)$ the maximum weight of an edge on $p_F(e)$, with the convention that $w_F(e) = \infty$ if v and u are not connected in F. We say e is F-*heavy *if $w(e) > w_F(e)$ and F-*light *otherwise. If $f \in F$ then we say f is F-*heavy *if there exists $e \notin F$ so that $f \in p_F(e)$ and $w(f) > w(e)$.*

The following lemmas are trivial folklore properties of MST's, we include their proof for completeness.

Property. 1 Cycle property *The heaviest edge in any cycle cannot be in the minimal spanning forest.*

Proof: Let e be the heaviest edge in some cycle C. Assume T is a MSF and $e \in T$. Let $U, E \setminus U$ be the two connected components in $T \setminus \{e\}$. Since C is a cycle, there are two edges in C in the cut $(U, E \setminus U)$ one which will be e. Let $f \in C$ be the other edge in $(U, E \setminus U)$. Since e is of maximal weight, $w(e) > w(f)$. The forest $T' = T \cup \{f\} \setminus \{e\}$ is a spanning forest with $w(T') < w(T)$. In contradiction to the minimality of T. ■

Property. 2 Cut property *The lightest edge in any cut must be in the minimal spanning forest.*

Proof: Let $e = (u, v)$ be the lightest edge in some cut C. Assume T is a MSF and $e \notin T$. Since u and v are in opposite sides of the cut there must be an edge $f \in T$ on the path between them so that $f \in C$. Since e is lightest in C, $w(e) < w(f)$. The forest $T' = T \cup \{e\} \setminus \{f\}$ is a spanning forest with $w(T') < w(T)$. In contradiction to the minimality of T. ■

Therefore only light edges can possibly belong to the MST.

2.3 Boruvka Step

The first MST algorithm was devised by Boruvka [2], the algorithm is based on the cut property. The basic step in Boruvka's algorithm is the heart of many MST algorithms till today. We therefore present this basic step and algorithm.

Boruvka Step: For each vertex v, select the minimum-weight edge incident to v. Contract all the selected edges, replacing by a single vertex for each connected component defined by the selected edges. Such a contracted component will be termed a *meta vertex*. Delete all resulting isolated vertices, loops (edges whose endpoints are both the same), and all but the lowest-weight edge among each set of multiple edges.

By the cut property applied to the cut $(\{v\}, V \setminus \{v\})$ the edges we contracted belong to the minimal spanning tree. A single Boruvka step requires time which is linear in the number of edges, and the step reduces the number of vertices in the graph by at least a factor of two. After no more than $\log n$ Boruvka steps we are left with a single vertex, the minimal spanning tree is the set of edges we contracted in this process. Therefore the Boruvka algorithm runs in time $O(m \log n)$

Another simple algorithm is the **Greedy Algorithm**. Given the edges in ascending order of weight, we define the graph $G^{(i)}$ to be a graph containing only the edges $e_1, \ldots, e_i$. An edge e_i belongs to the minimum spanning tree iff it is adjacent to two different connected components in $G^{(i-1)}$. Thus we add the edges one by one in ascending order, and the correctness follows from the cut property. Sorting the edge weights cost $O(m \log m)$, but once the edges are in ascending order of weight the algorithm takes linear time to find the MST.

2.4 Deterministic Algorithms

The MST problem has a long history. The first algorithm was discovered in 1926 by Boruvka [2], the algorithm is described in section 2.3 and has complexity $O(m \log n)$. This was first improved by Yao [24] to $O(m \log \log n)$. In the past two decades faster and faster algorithms were found beginning with Fredman-Tarjan [6] lowering the complexity to $O(m\beta(m, n))$ using Fibonacci heaps, where $\beta(m, n)$ is the number of log iterations on n needed to make it less than $\frac{m}{n}$. Shortly after, Gabow et al. [7] reduced the complexity to $O(m \log \beta(m, n))$. Recently Chazelle [3] has given an algorithm with complexity $O(m\alpha \log \alpha)$ where $\alpha = \alpha(m, n)$ is the inverse of Ackermann's function.

All the algorithms cited above use the Boruvka step, all search for the minimal edge incident to a meta vertex, and contract on it. The improvements in the running time are due to more complex and efficient data structures that quickly select this minimal edge. Chazelle's algorithm differs slightly from this greedy approach with his data structure allowing a controlled amount of error which is later fixed.

For all but very sparse graphs most of the algorithms mentioned achieve an optimal linear running time, but for graphs with $m \leq n \log^* n$ there is still a gap between the performance of these algorithms and the trivial linear time lower bound.

2.5 MST Verification

The cut property identifies MST edges, making its use dominant in the MST algorithms of section 2.4. The cycle property identifies edges which are not in the MST. We show how this property can be used for MST verification.
VMST Problem: *The problem of* MST verification *is as follows:*
Input: *A weighted graph* (G, w) *and a candidate tree* T *spanning* G.
Output: *Verify if* T *is the MST of* (G, w) *and if not identify the F-heavy edges in* T*, which prevent* T *from actually being the MST.*

VMST can be achieved in deterministic linear time by a recent algorithm due to King [13] based on ideas of Komlos [14] and using least common ancestor (LCA) queries.

We now sketch the verification algorithm: The algorithm verifies that each non-tree edge e is the heaviest in the cycle $cyc_T(e)$.

```
VMST($G$,$T$) {
    $T'$ = Build-Boruvka-Tree($T$)
    LCA = Build-LCA($T'$)
    Query = Komlos-Preprocessing($T'$)
    $BAD = \emptyset$
    **foreach** $(u, v) \in G \setminus T$ {
        $p = LCA(v, u)$
        $w_{max}$ = max(Query($u, p$),Query($v, p$))
        **if** $(w(u, v)) < w_{max})$
            add $(u, v)$ to $BAD$
    }
    **return** $BAD$
}
```

The set BAD contain all the F-heavy edges in T, if BAD is empty, T is the MST.

Definition 2 *A* full branching tree*, is a tree in which each non-leaf has at least two children and all leaves are at the same depth.*

A Boruvka-Tree is a full branching tree that describes the process of the Boruvka algorithm on T: T''s leaves are T's vertices and all other vertices correspond to the meta-vertices formed by the boruvka steps. The nodes at height i all correspond to the meta-vertices in the tree after the application of i Boruvka steps to T. An edge in $(a, b) \in T'$ corresponds to the edge e that was contracted vertex a into the meta vertex b and has the same weight $w(e)$. The maximal weight on any path between two leaves in T' (that correspond to u and v in T), is the same as $w_T((u, v))$. For further details see [13]. T' is at most twice the size of T and its construction takes linear time.

The Komlos-Preprocessing creates a table that for each vertex $v \in T'$, contains the heaviest edge in each of paths from v to all of its ancestors. Komlos [14] shows how to construct such a table for full branching trees in linear time. After such preprocessing, for any edge e, to query $w_T(e)$ takes $O(1)$ time.

3 Reduction to 3-Regular Graphs

As noted in section 2.4 the instances which are "hardest" for the known deterministic algorithms are those of linear size. This can be intuitively explained as follows: each of the known algorithms search for the $n-1$ edges that are in the MST. The denser the graph, the more edges we have to charge for this search. We now show that these graphs are inherently the hardest cases for the problem. We do this by showing that if we had a linear time algorithm for a 3-regular graph, we can devise a linear time algorithm for any graph, i.e. linearly reducing the general MST problem to its restriction for 3-regular graphs.

Let A be a linear time algorithm for 3-regular graphs, let G be a graph. We solve MST for G as follows:

```
Gen-MST($G$) {
    $G'$ = Transform-To-3REG($G$)
    $T' = A(G')$
    **return** Compute-Original-Tree($T'$,$G$)
}
```

We now describe Transform-To-3REG, for a vertex v:

- If $d(v) = 1$: Remove v from G.
- If $d(v) = 2$: Let $e_1 = (v, u)$ and $e_2 = (v, w)$ be the edges adjacent to v, w.l.o.g. $w(e_1) < w(e_2)$. Replace v with a single edge $e = (u, w)$, setting $w(e) = w(e_2)$.
- If $d(v) = k$ for $k > 3$: Let $e_1 = (v, u_1), \ldots, e_k = (v, u_k)$ be the edges incident to v. Replace v with a cycle of k vertices $v_1, \ldots, v_k$. Set the weights of the new edges $(v_i, v_{i+1} \text{mod} k)$ to be $-\infty$ Replace each e_i with (u_i, v_i) maintaining $w((u_i, v_i)) = w(e_i)$.

We now describe Compute-Original-Tree:

- W.L.O.G the edge weights in G are distinct. For each $e' \in T'$, if $w(e') \neq -\infty$, add to T the edge $e \in G$ so that $w(e) = w(e')$.
- For each $v \in G$ so that $d(v) \leq 2$.

- If $d(v) = 1$: Add to T the single edge incident to v.
- If $d(v) = 2$: Let $e_1 = (v, u)$ and $e_2 = (v, w)$ be the edges adjacent to v, w.l.o.g. $w(e_1) < w(e_2)$. If there is $e \in T'$ so that $w(e) = w(e_2)$ add both e_1 and e_2 to T, otherwise add only e_1 to T.

Lemma. 3 *The existence of a linear time MST algorithm for a 3-regular graph, implies a linear time algorithm for any graph.*

Proof: The algorithm Gen-MST computes the MST of any graph G in linear time. Transform-To-3REG is a linear time reduction from a graph $G(n, m)$ to $G'(m, 2m)$. Computing T from T' is simple becuase G is a minor of G', therefore by the cut property the edges of G whose corresponding edges are in T' belong in T as well. Since all the edges between $(v_i, v_{i+1} \text{mod} k)$ are lighter than all edges of G, all paths in T' between v_i and v_j will be only through these edges. Each of the steps Gen-MST: Transform-To-3REG, A, and Compute-Original-Tree take linear time in their input size. Notice that G' is at most twice the size of G, because for each edge in G we add at most one new edge in G'. ■

Note that a special algorithm for 3-regular graphs can not be based on edge contractions, this is because the contraction operation does not preserve regularity, or even a bounded degree in the vertices.

We will use this lemma in a future section.

4 Randomization and MST

In 1995 Karger et al. [12] gave a randomized algorithm that runs in $O(m)$ expected time. The algorithm uses random sampling on the edges along with a MST verification algorithm. This algorithm uses both the cycle and the cut properties in a fundamental way. It uses a randomly sampled fraction of the edges to obtain an approximated solution. Then using MST verification, finds the misplaced edges and recalculates the true MST in linear expected time.

4.1 Sampling Lemma

At the heart of the random algorithm is a sampling lemma.

Lemma. 4 *[12] Let H be a random subgraph obtained from G by omitting each edge independently with probability $\frac{1}{2}$, and let F be the minimum* *spanning forest of H. The expected number of F-light edges in G is at most $2n$.*

We present our proof of lemma 4. We believe it is a simpler and more informative proof than the one given in [12]. We present the notations for our proof. Assume the edge weights are $w_1 < w_2, \ldots < w_m$. We perform the random sampling sequentially, using fair independent coin tosses, we choose e_i if $\sigma_i = 1$. Let $\sigma \in \{0,1\}^m$ be the result vector of these coin tosses, with $\sigma^{(i)}$ denoting its i-prefix. Denote $G_{\sigma^{(i)}}$ the random graph that contains only the edges sampled from $e_1, \ldots e_i$ using $\sigma^{(i)}$.

Definition 3 *The* Sampled Connectivity Function *of a graph G, $f_G : \{0,1\}^i \rightarrow \{0,1\}$ for $i < m$, is a boolean function. $f_G(\sigma^{(i)}) = 1$ if e_{i+1} connects two disconnected components in the graph $G_{\sigma^{(i)}}$, otherwise $f_G(\sigma) = 0$.*

The sampled connectivity function indicates whether an edge e_{i+1} is a candidate for the MST in sequential sampling of G, namely would e_{i+1} be chosen by the greedy algorithm of section 2.3 for the graph $G_{\sigma^{(i)}}$. In other words if F is the spanning forest for $G_{\sigma^{(i)}}$, then $f_G(\sigma^{(i)}) = 1$ if e_{i+1} is F-light. If e will be chosen for the MST of $G_{\sigma^{(i-1)}}$ it will also be chosen for the MST of G_σ since only edges heavier that e_i are added to $G_{\sigma^{(i-1)}}$. Note that if F is the spanning forest of the sampled graph H then $\sum_{i=0}^{m-1} f_G(\sigma^{(i)})$ counts the number of F-light edges in G. In this terminology lemma 4 is as follows:

Lemma. 5 *Let G be a graph, for random $\sigma \in_U \{0,1\}^m$:*

$$E_\sigma \sum_{i=0}^{m-1} f_G(\sigma^{(i)}) \leq 2(n-1)$$

Proof: Note first for every G and every σ we have

$$\sum_{i=0}^{m-1} (f_G(\sigma^{(i)}) \wedge \sigma_{i+1}) \leq n-1 \tag{1}$$

Equation 1 is true because there are $\leq n-1$ edges in a spanning forest, and each edge that is both a candidate after the ith iteration and chosen at the $i+1$th iteration becomes part of the MSF. By using equation 1 we now prove the lemma.

$$E_\sigma[\sum_{i=1}^{m} f_G(\sigma^{(i)})] \quad = \quad \sum_{i=1}^{m} \Pr_\sigma[f_G(\sigma^{(i)}) = 1] = \tag{2}$$

$$= \quad 2\sum_{i=0}^{m-1} \Pr_{\sigma}[(f_G(\sigma^{(i)}) = 1) \wedge (\sigma_{i+1} = 1)] = \qquad (3)$$

$$= \quad 2\sum_{i=0}^{m-1} E_{\sigma}[f_G(\sigma^{(i)} \wedge \sigma_{i+1}] \leq 2n \qquad (4)$$

The transition between (2) to (3) is due to the independence of σ_{i+1} from $\sigma^{(i)}$. Since σ are fair coin tosses: $\Pr_{\sigma}[(f_G(\sigma^{(i)}) = 1) \wedge (\sigma_{i+1} = 1)] = \Pr_{\sigma}[(f_G(\sigma^{(i)}) = 1)] \Pr_{\sigma}[(\sigma_{i+1} = 1)] = \frac{1}{2} \Pr_{\sigma}[(f_G(\sigma^{(i)}) = 1)$ ■

From this proof it is clear why the random sampling works. Another benefit is that it brings forward the importance of independence. One may consider sampling using any deterministic scheme, or any standard probability distribution for derandomization such a k-wise independence [18], but it seems any such attempt is not enough to "cheat" this lemma. If we can predict a future coin, we simply construct a graph so that $f_G(\sigma^{(i)}) = \neg\sigma_{i+1}$ as many times as possible.

4.2 Randomized Algorithm

Using the sampling lemma and VMST a randomized algorithm for MST is as follows: if we perform verification for G on the MSF of a sampled graph H, less than $2n$ edges $e \in G$ are expected to satisfy either $e \in F$, or e connects dis-connected components in F, or $e \notin F$ but e is not maximal in $cyc_F(e)$. VMST finds these edges in linear time.

Here is the sketch of the random algorithm of [12]:

```
RMST($G$) {
    **if** ($G$ is small)
        return MST-By-Brute-Force($G$)
    ($G'$,$T'$) = Boruvka-Step($G$,2)
    $H$ = Sample($G'$,$\frac{1}{2}$)
    $F$ = RMST($H$)
    $G''$ = VMST($G$,$F$)
    $T''$ = RMST($G'' \cup F$)
    **return** $T' \cup T''$
}
```

The algorithm is based on a double recursion: On one hand, sampling ensures that the expected size of H for first recursive call is small ($\frac{m}{2}$). On the other hand, the sampling lemma 4 ensures that the expected size of G'' for for the second recursive call is small ($2n$). The algorithm's recursion relation is $A(m) = A(\frac{m}{2}) + A(2n)$, from this relation it is clear that the sampling is not effective when the number of edges is nearly equal to number of vertices. Therefore the algorithm performs two Boruvka steps on the graph returning the contracted graph G' and the contracted edges T'. Two Borurvka steps reduce the number of vertices in G' by at least a factor of four. After the Boruvka steps, Sample takes a random sampling of G''s edges: Each edge is sampled independently with probability $\frac{1}{2}$. To summerize Sampling reduces the number of edges by a constant factor and Boruvka step reduces the number of vertices by a constant factor.

The expected running time of the algorithm is $O(m)$. In fact the algorithm runs in $O(m)$ time with probability $1 - e^{-\Omega(m)}$. The worst case running time is $O(\min\{n^2, m \log n\})$, the same bound as for Boruvka's algorithm.

The existence of such an elegant linear time algorithm for MST in the randomized computational model, emphasizes the importance of finding a linear time deterministic algorithm. Does randomization really help, and only because of randomization is the complexity of MST linear? Or does there exist a deterministic linear time algorithm? Either result would be extremely interesting.

5 A Poset Framework

In a more powerful model, where bit manipulations are allowed on the edge costs, Fredman and Wilard [5] have devised a linear time algorithm. Comparing edge weights seems to encompass much of the combinatorial hardness of the MST problem: all the deterministic algorithms described in section 2.4 not only use a non-linear number of operations, they all use a non-linear number of edge comparisons as well. Furthermore, by edge comparisons we acquire information on the linear order of the edge weights. Had the linear order of the edges been given explicitly as part of the input, the problem would have easily been solvable by a linear time greedy algorithm. The question that arises is how many comparisons are needed to determine the MST. This naturally leads us to the comparison tree computational model.

In the previous chapters w was defined as a function that gives each edge an unique weight. These edge weights define a linear order on the edges:

$e_i <_w e_j$ if $w(e_i) < w(e_j)$. In this section we will sometimes relate to w as the linear order it induces, whether w relates to a function or linear order will be clear from the context.

Definition 4 *A* poset *is a set of constraints on the relations of the elements in E of the type $x < y$.*

Definition 5 *If P and Q are posets on a set E, we say that Q* extends *P if Q contains all the relations of P and possibly some others.*

We denote $L(P)$ the set of all linear extentions of P.

Definition 6 *The* height *of a poset is the length of the longest chain in the poset.*

5.1 Comparison Tree Model

In the comparison tree model the only resource we charge for is a comparison between two elements from an element set E. An algorithm in this model is a decision tree, D. We begin the computation at the root of the decision tree, at this point the linear order w on E is completely unknown. Each vertex in D corresponds to a query comparing between two elements $x : y$. We move down to the left or to the right child depending on the answer. As we advance down the decision tree, in node v we know w is an extension of P_v, the poset defined by the answers to our queries along the path to v. Each leaf represents a different answer to the computation. The leaves of the decision tree correspond to posets that partition the space of all linear extensions. The complexity of a decision tree is its depth. Therefore if there are k possible answers, the information theoretic lower bound on the depth of D is $\log k$.

Fredman [4] formulated a framework for sorting problems in the comparison tree model. A problem in this framework is defined by a pair (Π, P), where Π is a subset of the possible linear orderings on the weighted elements $w_1, \ldots, w_m$, and P is a partition of Π into disjoint sets $T_1, \ldots, T_k$. Given some instance $\pi \in \Pi$, the problem (Π, P) calls for determining to which set T_i, π belongs.

Many basic sorting problems have been extensively studied in this framework, including sorting, partial sorting, selecting an element of prescribed rank, and shortest paths [11, 15, 10, 22]. The MST problem naturally fits into this framework as well. However, to our knowledge, no one has addressed the MST problem in this model.

We can relax the MST problem to this model as follows: Fix a graph G, so the input is only the edge weights. Construct a decision tree which may depend on G (the preprocessing time for this construction is unlimited). Given as input a set of edge weights, compute the MST using the decision tree.

Denote the number of possible spanning trees for a graph G: $N_T(G)$. The information theoretic lower bound on the depth of such a tree is $\log N_T(G)$. It should be noted that $N_T(G)$ is less than $\binom{m}{n-1} = 2^{O(n+m)}$, hence the information theoretic lower bound is a linear number of comparisons.

5.2 Reduction of MST to Posets

The edge weights in G define a linear ordering w on E. This linear order implies a unique minimum spanning tree. We present an idea that facilitates the other viewpoint of this implication, i.e. that each spanning tree T defines a poset P_T that uniquely characterizes T.

Let Ψ_T be the set of all linear orders w of E so that T is the MST for (G, w).

Property. 6 *There exists a poset P_T such that $w \in \Psi_T$ iff p is a linear order extending P_T.*

This correspondence is based on the cycle property, using ideas similar to verification of MST. We define the relations in the poset P_T that spans T.

$$P_T = \bigcup_{f \notin T} \{f > e | e \in cyc(f)\}$$

Definition 7 *In such a case we call P_T the* Spanning Poset *for the tree T.*

Since the cycle property is a necessary and sufficient condition for T to be a MST. T is the MST for each linear order $w \in L(P_T)$ and visa versa. Denote $\mathcal{P} = \{P_T | T \text{ is a spanning tree for } G\}$ to be the set of all spanning posets of G. $\mathcal{P}$ is a disjoint partition of $\mathcal{S}_m$: Each permutation of E is a linear extension of exactly one of the spanning posets.

Each element in P_T corresponds to an edge in the graph G. Observe some of the following characteristics of spanning posets:

- The height of these posets is always one.
- The minimal elements correspond to the tree edges and there are exactly $n - 1$ such elements.

- The maximal elements are the non-tree edges and there are $m - n + 1$ such elements.
- Each non-tree edge e is greater than all the tree edges $cyc(e)$.
- The cut property which is dual to the cycle property leads to the same spanning poset.

Using this correspondence between spanning trees and posets, the problem of finding the MST in the decision tree model reduces to recognizing which spanning poset a given linear ordering extends. As we advance down the decision tree, in node v we know w is an extension of P_v, the poset defined by the answers to our queries. We say that a tree T is feasible at node v if there exists a poset Q so that Q extends both P_v and P_T. The leaves of the decision tree all correspond to posets that extend a unique spanning poset.

5.3 Inside the Poset Framework

In a decision tree, which is ideal from the information theoretic standpoint, each spanning poset P_T has exactly one leaf in the decision tree that extends P_T. However this is not the case i.e., there must be some spanning posets that have a number of different leaves that extend them. This follows from each query having spanning trees that remain feasible MSTs for either answer to the query. This happens, for instance, when the queried edges are both tree edges or both not tree edges. For such trees the order between the queried edges does not matter.

This framework allows us to naturally characterize questions such as: What spanning trees does the comparison $e < f$ disqualify from set of feasible MSTs extenting P. A spanning tree T is disqualified if the Hasse diagram for the poset $P_T \cup P \cup \{e < f\}$ contains a directed cycle.

Note that reaching the information theoretic lower bound of $\log N_T(G)$ is not possible. The problem of finding a maximum of n elements is a special case of MST, since solving MST for a simple cycle is deciding the maximal element on that cycle. Finding the maximum in a set of m elements requires m comparisons, not $\log m$. Therefore a minimal reasonable lower bound on the depth of the decision tree is $O(n + m)$.

In [11] it is shown that in every finite poset there are elements x and y, whose comparison reduces the number of linear extensions by a constant fraction, regardless of the outcome of the comparison. For our problem the analogue is: For every poset, does there always exist two edges so that their comparison reduces the number of possible spanning trees by a constant fraction? Unfortunately the answer to this question is negative. Consider the following example: The graph G is a wheel. G has a center vertex u and the vertices $v_1, \ldots, v_n$. The edges of G are the edges of the star (u, v_i) and the edges of the cycle $(v_i, v_{(i+1)\mathrm{mod} n})$. In the poset P all the edges (v, u_i) for even i are minimal edges. Each such minimal edge is smaller than all non-minimal edges. For this poset any edge comparison can reduce at most a fraction $O(\frac{1}{n})$ of possible spanning trees.

Formalizing the MST problem as a problem in partial orders opens the possibility of using some of the many strong tools used to solve other algorithmic recognition problems in partial orders like [11, 15]. In order to be able to use such tools it is important to first understand the characteristics of these spanning posets and how they differ from other posets. We believe that a deeper understanding of these posets and how they partition $\mathcal{S}_m$ can lead to a constructive linear decision tree for MST.

6 An Algorithm for Small Separators

We present two divide and conquer algorithms for graphs with small separators that find a MST using only $O(m)$ comparisons. Since Lipton and Tarjan [16] first showed the existence of small vertex separators ($O(\sqrt{n})$ for planar graphs, separator theorems have been found for many families of graphs, and many algorithms have been proposed for such graphs, especially in the field of computational geometry. In the literature, it is more common to deal with vertex separators [16, 1, 21] while our algorithm requires edge separators. We show that for our purposes these are equivalent.

6.1 Definitions

We present algorithms that give correct answers for all graphs, and for graphs with small separators use only $O(m + n)$ comparisons. We now define graph separators and the properties required by our algorithms for linear time complexity. Let $G = (V, E)$ be a graph, $f : \mathcal{N} \rightarrow \mathcal{N}$ a function. We set a constant $\alpha = \frac{1}{10}$, ($\frac{1}{10}$ is arbitrary).

Definition 8 *$S \subseteq E$ is an f-*separator *if:*

1. $|S| < f(m)$

2. $V = V_1 \cup V_2$, with all edges between V_1 and V_2 being in S.

3. Each induced subgraph $G[V_i]$ has at least $\lfloor \alpha m \rfloor$ edges.

(compare with [21] for the definition of vertex separator)

We call the induced subgraphs $G_i = G[V_i]$ the graphs *induced* by the separator.

Definition 9 *A graph G is f-*recursably separable *if G has a f-separator and each of the subgraphs G_i induced by the separator are f-recursably separable.*

A *recursive separation* in a f-recursably separable graph is the set of all separators hierarchically created by the definition, until all edges are in some separator.

Definition 10 *A graph G is f-*strongly separable *if each subgraph G' of G has a f-separator.*

Definition 11 *A family of graphs $\mathcal{F}$ is called* monotone *if it is closed to the subgraph operation. Meaning if $G \in \mathcal{F}$ than for any $H \subset G$, $H \in \mathcal{F}$*

6.2 General Strategy

We present two algorithms for the MST problem. For any $\epsilon > 1$, the first runs in linear time for any G which is $\frac{m}{(\log m)^{2+\epsilon}}$-recursably separable, the second runs in linear time for any G which is $\frac{m}{(\log m)^{1+\epsilon}}$-strongly separable. Our algorithms differ from their predecessors in a fundamental way: While previous deterministic algorithms are all based on the cut property, our algorithms are based on the cycle property. We identify edges that are not in the MST and remove them from the graph. Our algorithms use a Divide and Conquer paradigm using graph separators.

We describe the general strategy of our algorithm. We begin with a graph G with separator S of size $f(m)$. If G is small enough, we find the MSF using exhaustive search. Otherwise we remove S. We are left with G_1 and G_2, where the number of edges in each subgraph is a constant fraction of the number of edges in the whole graph. We continue recursively on G_1 and G_2, eventually having T_1 and T_2 the MSF's of G_1 and G_2 along with the edges from S which we have not yet handled. Finally we update the forests of G_i, taking edges from S into consideration.

In order to efficiently implement this strategy, we need a method to efficiently handle the edges from the separator S. In order to perform this task we first need to characterize the candidates from S that can possibly belong to the MSF. Note that we not only need add the minimum edges that join unconnected components, there may be other edges in S that belong in the MSF. The edges of the MSF of $T1 \cup T2 \cup S$ can simply be characterized, by the cycle property, as those edges that are not the heaviest on any cycle in $T1 \cup T2 \cup S$. The two algorithms we present differ in the method in which we handle the separator edges.

Notice that the separator in the graph G is independent of the edge weights and finding S has no cost in the comparison model.

6.3 Algorithm using Dynamic Trees

We present an algorithm based on dynamic trees. We check edges in the separator, one by one, whether they need to be added to the forest or not. In order to manipulate the edges efficiently, we use the Sleator-Tarjan [23] dynamic forest structure, which allows us to manipulate each $e \in S$ in $O(\log n)$ steps (comparisons).

Algorithm:

```
MSF($G$) {
    $F_1$ = MSF($G_1$)
    $F_2$ = MSF($G_2$)
    **return**(update-MSF($F_1$,$F_2$,$S$))
}
```

```
update-MSF($F_1$,$F_2$,$S$) {
    $F = F_1 \cup F_2$
    **foreach** $(e \in S)$
        **if** ($e$ joins two distinct components in $F$)
            add $e$ to $F$
        **else**
            $f = max[cyc_F(e)]$
            **if** $(w(e) < w(f))$
                remove $f$ from $F$ and add $e$ to $F$
    **return** $F$
}
```

The Sleator-Tarjan data structure supports the following operations on rooted trees in only $O(\log n)$ steps:

- $root(v)$ - returns the root of the tree containing the vertex v
- $link(v, u, w)$ - combine the trees containing v and u by adding the edge (v, u) of cost w, making v and u roots of the corresponding trees and making v the parent of u.
- $cut(v)$ - divide the tree containing the vertex v into two trees by deleting the edge $(v, parent(v))$.
- $lca(v, u)$ - returns the lowest common ancestor of v and u assuming v and u are in the same tree.
- $maxweight(v, x)$ - returns edge of maximum weight along the path connecting v and x, where x is an ancestor of u.
- $weight(v)$ returns the weight of the edge $(v, parent(v))$.

We now present the algorithm in more detail:

```
update-MSF($F_1$,$F_2$,$S$) {
    $F = F_1 \cup F_2$
            // $F_1$ and $F_2$ each are Sleator Tarjan forests//
    **foreach** $((a,b) \in S,\ )$
        **if** $(root(a) \neq root(b))$
            $link(a, b, w((a,b)))$
        **else** {
            $c = lca(a,b)$
            $d = \max(maxweight(a,c), maxweight(b,a))$
                    //the edge of maximum cost on the path between $a$ to $b$//
            **if** $(w((a,b)) < w(d))$ {
                $cut(d)$
                $link(a, b, w((a,b)))$
            }
        }
}
```

Since for each edge we perform a constant number of operations, each with a cost of $O(\log m)$ steps, the complexity of the update stage is $O(|S| \log m)$.

Correctness: We prove the correctness of the algorithm by induction on the number of edges. The algorithm is based on removing non-MST edges from the graph using the cycle property. By the cycle property, any edge $e \in G_i$ so that $e \notin T_i$ can not be in the MST of G, since these edges are heaviest in some cycle in $G_i \subset G$. The update stage checks all not tree edges and uses the cycle property again to remove non-MST edges. Our algorithm terminates when we are left with a spanning tree. Since we removed from the graph only edges that are not in the MST, the tree that is left must be the MST.

Time Analysis: Assume G is $(\frac{m}{\log^{2+\epsilon} m})$-recursably separable. This would give the following recursion:

$$A(m) = A(\alpha m) + A((1-\alpha)m) + O(\frac{m}{\log^{2+\epsilon} m} \log m)$$

$$A(5) = O(1)$$

using lemma 8 this gives the linear running time.

6.4 Algorithm using Steiner Tree

We present another divide and conquer MST algorithm based on small edge separators. This algorithm is similar to the one described in figure 6.3, the main difference being in the method used in the update-MSF stage. The method in this algorithm is based on building a smaller graph G' so that after recursively solving MSF for G', we can construct the MSF of G without needing any more edge comparisons. We begin with a few definitions.

Definition 12 *Given a spanning tree T and a set of vertices $U \subset V$, the* Steiner tree *T_U is a tree defined on the vertices $U \cup L$ where $L \subset V$ is: $L = \{v | \exists x, y, z \in U \text{ s.t. } v \in P_T((x,y)), v \in P_T((x,z)), v \in P_T((y,z))\}$. There is a 1-1 correspondance between the edges in T_U and paths in T whose endpoints are in $U \cup L$ and inner nodes not in $U \cup L$. (such a path can be only one edge if $v, u \in U \cup L$ are neighboors in T.*

Definition 13 *A* Steiner forest *is a forest of Steiner trees on each connected component.*

We now define how edge weights for T_U are derived from T, this is based on the ∞-norm.

Definition 14 *Given a weighted spanning tree T with edge weights w, the* max-norm Steiner tree *on $U \subset V$ is the Steiner tree T_U with the following edge weights: Each edge $e \in T_U$ corresponds to a path P_e in T: $w(e) = \max_{f \in P_e} w(f)$.*

Constructing the Steiner tree is independent of the edge weights. Some edge comparisons are necessary in order to set the edge weights in the max-norm Steiner tree, but only a linear number of comparisons are needed. This is because all that is required is the maximum edge in each path for a set of disjoint paths in T.

Lemma. 7 *Let $|U| = s$, then the max-norm Steiner tree on U, T_U, has no more than $2s$ edges.*

Proof: This is because T_U is a tree on $U \cup L$ and has $|U \cup L| - 1$ edges. For any $v \in L$, $d(v) \geq 2$ and therefore $|L| \leq |U|$. ■

Definition 15 *Given a weighted graph G with weight function w and $U \subset V$ we define the w-*max-norm Steiner forest *$F_{(G,U,w)}$ as the max-norm Steiner forest of the minimal spanning forest of G given w.*

For a graph G with a separator S, vertices that are endpoints of a separator edge will be called *marked* vertices. As described in section 6.3 a separator S in G gives us the graphs G_1 and G_2. At this point we can describe the smaller graph G'. We set U to be the marked edges of separator S. Given the edge weights w, G' is the w-max-norm Steiner forests of G_1 and G_2 along with S.

$$G' = S \cup F_{(G_1,U,w)} \cup F_{(G_2,U,w)}$$

In order to perform the required updating, it suffices to know the heaviest edge on any path, in G, between two marked vertices. By definition, any edge representing the heaviest edge on any path between vertices from U will be contained in the w-max-edge Steiner forest. Therefore constructing the MSF of G is reduced to constructing the MSF of G'.

Algorithm:

```
MSF($G$) {
    $F_1, F_{G_1,U,w}$ = MSF($G_1$)
    $F_2, F_{G_2,U,w}$ = MSF($G_2$)
    $F', F_{G',U,w}$ = MSF($G'$)
    $F = F' \cup (F_1 \setminus F_{(G_1,U,w)}) \cup (F_2 \setminus F_{(G_2,U,w)})$
    $F_{G,U,w}$ = Construct-Steiner($F$)
    **return**($F$,$F_{G,U,w}$)
}
```

The MSF procedure not only constructs the minimal spanning forest F, it constructs the max-edge Steiner forest $F_{(G,U,w)}$ as well. Since $F_{(G_1,U,w)} \cup F_{(G_2,U,w)}$ have representatives for all the heaviest edges in any cycle in $\tilde{G}$, it is enough to recursively compute the MSF, F', in G'. Denote $s = |S|$, $s \leq f(m)$, G' is a graph smaller than G, containing only the separator and the max-edge Steiner trees. Hence, by lemma 7 G' is of total size $O(s)$. The max-edge Steiner tree does depend on the edge weights, but as already mentioned it is trivial to construct it using a linear number of comparisons.

Correctness: We prove the correctness of the algorithm by induction on the number of edges using the cycle property. As in the algorithm of section 6.3 only non-MST edges are removed. All stages of this algorithm construct MSF's of subgraphs of G. An edge not in the MSF of a subgraph, by the cycle property is not in the MSF of G. Our algorithm terminates when we have obtained a spanning tree. Since we removed from the graph only edges that are not in the MST, the resulting tree must be the MST.

Time Analysis Notice that to perform the recursion on G' we first need to find an edge separator in G'. Since G' depends on the weights w we do not know in advance which separators are needed, we have to pre-calculate a separator for any subgraph that the algorithm could create. This is the reason that this algorithm has the stronger requirement that the graph be strongly separable. This algorithm uses regular trees (not Sleator-Tarjan), therefore the size constraint of the separator can be relaxed to $\frac{m}{(\log m)^{1+\epsilon}}$ which improves the maximal separator size over the algorithm of section 6.3 by a factor of $\log m$.

Assume the graph has a $\frac{m}{\log^{1+\epsilon} m}$-separator for any subgraph. This would give the following recursion:

$$A(m) = A(\alpha m) + A((1-\alpha)m) + A(\frac{m}{\log^{1+\epsilon m}})$$

using lemma 8 this gives the linear running time.

6.5 Recursion

Lemma. 8 *The following recursion is linear:*

$$A(m) = A(\alpha m) + A((1-\alpha)m) + f(m)$$

$$A(5) = O(1)$$

where

$$f(m) \leq \frac{cm}{\log^{1+\epsilon} m}$$

for some $\epsilon > 0$ and constant c.

Proof: Set $k = 1+\epsilon$, $\alpha = \frac{1}{10}$ and $\beta = \frac{9}{10}$.

$$A(m) = m + m \sum_{i,j|\alpha^i\beta^j m>5} \binom{i+j}{i} \frac{\alpha^i \beta^j}{(\log(\alpha^i\beta^j m))^k}$$

The first m in the sum is for the bottom level of the recursion. Set $i_{max} = \min_i\{\alpha^i m < 5\}$ and $j_{max} = \min_j\{\beta^j m < 5\}$, we change the ordering of the summation to go along diagonal of the type $\alpha^i\beta^j m$ =const. Set $j_M = \min_j\{\alpha^{M-j_M}\beta^{j_M}m > 5\}$.

$$m \sum_{M=0..j_{max}} \sum_{j=j_M}^{M} \binom{M}{j} \frac{\alpha^{M-j}\beta^j}{(\log(\alpha^{M-j}\beta^j m))^k} \leq$$

Since the denominator is smaller but stays positive this is less than:

$$m \sum_{M=0..j_{max}} \frac{1}{(\log(\alpha^{M-j_M}\beta^{j_M}m))^k} \sum_{j=j_M}^{M} \binom{M}{j} \alpha^{M-j}\beta^j \leq$$

The last summation is less than one, this makes the whole expression less than

$$m \sum_{M=0..j_{max}} \frac{1}{(\log(\alpha^{M-j_M}\beta^{j_M}m))^k}$$

We show that the last sum is constant. Set $A = \log\alpha$, $B = \log\beta$ and $L = \log m$ We split the sum into two ranges

$$\sum_{M=1}^{i_{max}} \frac{1}{(j_M(B-A)+MA+L)^k} + \sum_{M=i_{max}}^{j_{max}} \frac{1}{(j_M(B-A)+MA+L)^k}$$

For the first sum $j_M = 0$, the second $j_M = i_{max} - \frac{(M-i_{max})j_{max}}{j_{max}-i_{max}}$ So we get the following sums:

$$\sum_{M=1}^{i_{max}} \frac{1}{(MA+L)^k} + \sum_{M=i_{max}}^{j_{max}} \frac{1}{(\eta M+\zeta)^k} \leq$$

η and ζ are constants. Remember that L and i_max are $\Theta(\log m)$ therefore we can bound the sum by

$$\sum_{M=1}^{\infty} \frac{1}{(c\log m)^k} \leq C$$

for $k > 1$, c and C are constants. This gives a linear recursion as required.
■

6.6 Separators

Many separator results have been found for many families of graphs. There are some important families for which the existence of small separators have been proven, some of which are presented below. All the families of graphs listed below are monotone. Therefore the separators exist recursively as required by our algorithm.

- Lipton and Tarjan [16] proved any plannar graph has a $\sqrt{8n}$-separator. It should be noted that a linear time algorithm for MST in plannar graphs is already known (Matsui) [20]. This algorithm is based on the fact that planner graphs always have a vertex with a small cut.

- Graphs that can be embedded on a surface of bounded genus (Gilbert, Hutchinson and Tarjan) [9]

- Graphs with no h-clique as a minor, have an $O(h^{\frac{3}{2}}\sqrt{n})$-separator. (Alon, Seymour and Thomas) [1]

- Density Graphs are graphs that can be embedded in d-dimensions so that for each node v, the ratio between distance to the farthest node to v and the closest node to v is less than a constant. These graphs have a $O(n^{\frac{d-1}{d}})$ (Miller, Teng, Vavasis) [21]

- Overlap Graphs generalize some of the previous families. The main idea is if we embed the p points in d dimensions and define n closed balls so that each contains at most k points, an overlap graph is an intersection graph between the balls. (Miller, Teng, Vavasis) [21]

All the separator results listed above are for vertex separators. We show how they can still be employed for our algorithms which require edge separators. For graphs of constant bounded degree, it is obvious that a small vertex separator is equivalent to a small edge separator: We simply remove all edges adjacent to all vertices in the separator. Using lemma 3 this can be extended for any graph.

Separator results for monotone families of graphs lead to applicability of the divide and conquer algorithms presented. For such graphs the algorithm performs only a linear number of edge comparisons. This constitutes a very large family of sparse graphs for which we have a constructive linear decision tree algorithm. Some of the open questions that need to be asked at this point are:

- Is there a complementary algorithm for graphs in which all separators are large?
- Can this linear decision tree algorithm lead to a linear algorithm on pointer machines? Note that a fast sub-linear algorithm to find separators would give such a pointer machine algorithm.

7 Conclusion

This thesis asks more questions than it answers. In addition to the main positive result: a linear time algorithm for graphs with small separator in the decision tree model, we expressed many of the reasons that make the MST problem so hard. We still conjecture that the MST has a linear time deterministic complexity. We propose a new framework in which to study the MST which might lead to new results. The future directions which seem to us most promising are following:

- Further understand the MST in the poset framework. Does there exist a measure other than number of possible spanning trees that shrinks by some constant fraction with edge comparisons?
- Is there a linear depth decision tree for MST?
- Is there another simple combinatorial property aside from the cut and cycle property that drives the MST problem?
- Is there another linear randomized algorithm for MST? One that is conjectured in the decision tree model is using a uniformly chosen spanning tree. Perform verification on this tree discard heavy edges.

Acknowledgments

I would like to thank my advisor, Avi Widgerson, for showing me the magic of theory of computer science and having the patience to teach me. I would like to thank my husband, Itsik Pe'er, for cheering me up when I was stuck and discussing ideas with me at any hour of the day. I would like to thank my parents, Mara and Aaron Beller, for bringing me up with curiosity and the love of learning. I would like to thank Valerie King, Nati Linial and Mike Saks for many useful discussions. And I would like to thank the students at the HUJI theory lab for making studying together so enjoyable.

References

[1] N. Alon, P. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proc. 22nd Annual ACM STOC*, pages 293–299, 1990.

[2] O. Boruvka. O jistem problemu minimalnim. *Praca Moravske Prirodovedecke Spolecnosti*, 3:37–58, 1926. In czech.

[3] B. Chazelle. A deterministic algorithm for minimum spanning trees. In *FOCS*, page ????, Texas ???, 1997.

[4] M. Fredman. How good is the information theory bound in sorting. *Theoretical Computer Science*, 1:355–361, 1976.

[5] M. Fredman and D.E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In *FOCS*, pages 719–725, 1990.

[6] M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, 1987.

[7] H.N. Gabow, Z. Galil, T. Spencer, and R.E. Tarjan. Efficient algorithms for finding mimimum spanning trees in undirected and directed graphs. *Cominatorica*, 6:109–122, 1986.

[8] J.A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numerical Analysis*, 10:345–367, 1973.

[9] J. Gilbert, J. Hutchinsom, and R.E Tarjan. A separator theorem for graphs of bounded genus. *J Algorithms*, 5:391–407, 1984.

[10] R. Graham, F. Yao, and A. Yao. Information bounds are weak in the shortest distance problem. *JACM*, 27:428–444, 1980.

[11] J. Kahn and L Linial. Balancing extensions via brunn-minkowski. *Combinatorica*, 11:363–368, 1991.

[12] D.R. Karger, P.N Klein, and R.E. Tarjan. A randomized linear-time algorithms to find minimum spanning trees. *J. ACM*, 42:321–328, 1995.

[13] V. King. A simpler minimum spanning tree verification algorithm. *Manuscript*, 1995.

[14] J. Komlos. Linear verification for spanning trees. *Combinatorica*, 5:57–65, 1985.

[15] N. Linial. The information theoretic bound is good for merging. *SIAM J, Comp.*, 13:795–801, 1984.

[16] R.J. Lipton and R.E Tarjan. A separator theorem for planner graphs. *SIAM J of Applied Math*, 36:177–189, 1979.

[17] L. Lovasz. ?? *Discrete Math*, 13:383–390, 1975.

[18] A. Luby, M. Wigderson. Paiwise independence and derandomization. Lecture notes, 1995.

[19] Blum M., Floyd V., Pratt V., Rivest R., and Tarjan R. Time bounds for selection. *J. Computer Sys. Sci*, 7:448–461, 1973.

[20] T. Matsui. The minimum spanning tree problem on a planner graph. *Discrete Applied Mathematics*, 58:91–94, 1995.

[21] G.L. Miller, S. H. Teng, and S.A. Vavasis. An unified geometric approach to graph separators. In *FOCS*, pages 538–547, 1991.

[22] M. Saks. The information theoretic bound for problems on ordered sets and graphs. *Graphs and Orders*, pages 137–168, 1985.

[23] D.D. Sleator and Tarjan R.E. A data structure for dynamic trees. *J. of Computer and System Sciences*, 26:362–391, 1983.

[24] A. Yao. An $o(|e| \log\log |v|)$ algorithm for finding minimum spanning trees. *Inf. Process Lett.*, 4:21–23, 1975.