Using Sparsification for Parametric Minimum Spanning Tree Problems

David Fernández-Baca[1*], Giora Slutzki[1], and David Eppstein[2]

[1] Department of Computer Science, Iowa State University, Ames, IA 50011.
[2] Department of Information and Computer Science, University of California, Irvine, CA 92717.

Abstract. Two applications of sparsification to parametric computing are given. The first is a fast algorithm for enumerating all distinct minimum spanning trees in a graph whose edge weights vary linearly with a parameter. The second is an asymptotically optimal algorithm for the minimum ratio spanning tree problem, as well as other search problems, on dense graphs.

1 Introduction

In the parametric minimum spanning tree problem, one is given an n-node, m-edge undirected graph G where each edge e has a linear weight function $w_e(\lambda) = a_e + \lambda b_e$. Let $Z(\lambda)$ denote the weight of the minimum spanning tree relative to the weights $w_e(\lambda)$. It can be shown that $Z(\lambda)$ is a piecewise linear concave function of λ [Gus80]; the points at which the slope of Z changes are called *breakpoints*.

We shall present two results regarding parametric minimum spanning trees. First, we show that $Z(\lambda)$ can be constructed in $O(\min\{nm \log n, T_{MST}(2n, n) \cdot b(m, n)\})$ time, where $T_{MST}(m, n)$ is the time to compute a minimum spanning tree and $b(m, n)$ is the worst-case number of breakpoints of $Z(\lambda)$. It is known that $b(m, n) = O(m\sqrt{n})$ [Gus80, KaIb83] and $b(m, n) = \Omega(m\alpha(n))$ [Epp95], and that $T_{MST}(m, n) = O(m \log \beta(m, n))$ time [GGST86] (here $\beta(m, n) = \min\{i : \log^{(i)} n \leq m/n\}$) [3]. Our algorithm improves on the Eisner-Severance method [EiSe76], which, when applied to parametric minimum spanning trees, takes $O(T_{MST}(m, n) \cdot b(m, n))$ time. For example, if $T_{MST}(m, n) = O(m \log \beta(m, n))$ and $b(m, n) = O(m\sqrt{n})$, then our algorithm runs in $O(nm \log n)$ time, while the Eisner-Severance approach takes time $O(n^{1/2}m^2 \log \beta(m, n))$; the speedup is less dramatic, but still significant, if $b(m, n)$ is, say, $O(m\alpha(n))$. Our construction procedure can be used for problems such as the stochastic spanning trees studied by Ishii et al. [ISN81], which are solved by enumerating the different

* Supported in part by the National Science Foundation under grant No. CCR-9211262 and grant No. CCR-9520946.

[3] Linear time (i.e., $O(m)$) algorithms can be obtained either through randomization [KKT94], or by working in less restrictive models of computation [FrWi90].

solutions obtained when the parameter is varied throughout its range. Several other applications are discussed by Hassin and Tamir [HaTa89].

Our second result is an approach for solving the following closely-related parametric search problems in $O(n\sqrt{m}\log^2(n^2/m))$ time:

(P1) Given a value λ_1 find the first breakpoint λ^* of $Z(\lambda)$ such that $\lambda^* \geq \lambda_1$.

(P2) Find a value λ^* such that $Z(\lambda^*) = 0$. This assumes that the slopes of the edge weight functions are either all negative or all positive; we shall assume the former.

(P3) Find a λ^* such that $Z(\lambda^*) = \max_\lambda Z(\lambda)$.

Problem (P1) is a standard problem in sensitivity analysis [Gus83], (P2) arises in the solution of the *minimum-ratio spanning tree problem* (MRST) [Cha77, Meg83], and (P3) arises in Lagrangian relaxation [CMV89]. Our result should be contrasted with the fastest known algorithm for these problems, due to Cole [Cole87], which solves them in $O(T_{MST}(m,n)\log n) = O(m\log\beta(m,n)\log n)$ time. Thus, our algorithm is faster than Cole's for all sufficiently dense graphs — i.e., $m = n^2/o(\log n)$ — and is optimal, to within a constant factor, for graphs with $\Theta(n^2)$ edges. While our search procedure is an improvement over existing methods only for the upper ranges of graph density, it constitutes what, to our knowledge, is the first non-trivial progress on the problem in many years. We hope that this result will lead to improvements in other ranges as well.

Cole's parametric minimum spanning tree algorithm is a clever application of Megiddo's method of parametric search [Meg79, Meg83], a technique with numerous applications to optimization and computational geometry [CoMe93, Tol93a, CEGS92, MaSc93, ShTo94]. Indeed, MRST has the distinction of being one of the original problems to which Megiddo applied his approach. Megiddo's technique relies on the existence of an algorithm $\mathcal{E}$ for the underlying non-parametric version of the problem; for MRST, such an algorithm would solve the minimum spanning tree problem. Algorithm $\mathcal{E}$ is simulated to determine its computation path at the value λ^* being sought. Algorithm $\mathcal{E}$ is also used to implement an oracle for narrowing the search interval. This dual use of $\mathcal{E}$ tends to lead to algorithms that are, at best, a polylogarithmic factor slower than the algorithms for the underlying non-parametric problems.

The research reported here is part of an ongoing attempt to determine the conditions under which one can eliminate the slowdown that seems inherent in Megiddo's method. We have been motivated in part by Frederickson's observation [Fre90] that, as the search progresses, it is sometimes possible to compile information that can speed up subsequent oracle calls. This idea was used to devise linear-time algorithms for a variety of location problems on trees [Fre90]. Subsequently we showed that a large class of parametric optimization problems can be solved in linear time for graphs of bounded tree-width [FeSl94]. More recently, we showed that related techniques lead to linear time algorithms for several parametric minimum spanning tree problems on planar graphs [FeSl95]. The above-mentioned work and the results in the current paper suggest that a combination of certain ideas can lead to the elimination of much of the overhead implicit in Megiddo's method. Among these is the use of sampling to decide which portion of the search interval to focus our attention on, and the use of a looser variant of Megiddo's method in which, instead of attempting to maintain only one computation path of $\mathcal{E}$ at any phase of the simulation, several are maintained simultaneously; this exhibits the strong connection between construction and search problems. A crucial component of our algorithms is *sparsification* [EGIN92, EGI93], which allows us to organize computations in such a way that successive steps examine progressively less of the input. Finally, our algorithms rely critically on the processing of intersections of arrangements of lines associated with edge costs. We note that, in contrast to many of the known applications of parametric search, we do not rely explicitly on parallelism in the solution of the underlying non-parametric problem.

2 Sparsification

Sparsification relies on the observation that for many graph problems — minimum spanning trees included — the portion of the graph that actually participates in the final solution is relatively small. The same property often holds for updates as well, a fact that has been used by Eppstein et al. [EGIN92, EGI93] to solve several dynamic graph problems. We shall now summarize one of the key ideas behind sparsification: the use of *sparsification trees*, which were introduced in [EGI93] (see also Frederickson's work [Fre85a, Fre91]).

Sparsification trees are built in two steps. In the first, a *vertex partition tree* is constructed by splitting the vertex set into two equal-size parts (to within 1) and then recursively partitioning each half. This results in a complete binary tree of height at most $\log n$ where nodes at depth i have $n/2^i$ vertices. The vertex partition tree is used to build an *edge partition tree*. For any nodes α and β of the vertex partition tree at the same depth i, containing vertex sets V_α and V_β, we create a node $E_{\alpha\beta}$ in the edge partition tree containing all edges of G in $V_\alpha \times V_\beta$. The parent of $E_{\alpha\beta}$ is $E_{\gamma\delta}$, where γ and δ are, respectively, the parents of α and β in the vertex partition tree. An internal node $E_{\alpha\beta}$ will have three children if $\alpha = \beta$ and four otherwise. The *sparsification tree*, denoted T_G, is built from the edge partition tree by including only those nodes $E_{\alpha\beta}$ that contain at least one edge of $E(G)$.

Consider any node $u = E_{\alpha\beta}$ in T_G. We will denote by G_u the subgraph of G whose vertex set is $V_\alpha \cup V_\beta$ and whose edge set is $E(G) \cap (V_\alpha \times V_\beta)$. The next lemma summarizes, for future reference, properties of sparsification trees that are either straightforward or have been proved elsewhere [EGI93].

Lemma 1. *For any k between 0 and the depth of the sparsification tree, (i) there are at most 2^{2k} depth k nodes; (ii) the edge sets of the graphs associated with the nodes at depth k are disjoint and form a partition of $E(G)$; and (iii) if u is a node at depth k, $|V(G_u)| \le n/2^k$, and $|E(G_u)| \le n^2/2^{2k}$.*

3 Generating $Z(\lambda)$

We now discuss how to generate $Z(\lambda)$ in $O(mn \log n)$ time.

We assume without loss of generality that no two pairs of edge weights become equal simultaneously; this can be achieved without significantly changing $Z(\lambda)$ by perturbing the weight functions infinitesimally. Such a perturbation can be done symbolically without changing the asymptotic running time of our algorithms.

First let us discuss a simpler algorithm for the same problem, which runs in $O(m^2 \log n)$ time, and appears to be folklore. Our improved result comes from applying sparsification to this technique. The key observation (also used in previous work on the subject) is that if λ^* is a breakpoint of $Z(\lambda)$, then two edges e and f have equal weights at λ^*, and the symmetric difference of the minimum spanning trees before and after λ^* is $\{e, f\}$. The idea is to compute for each pair $\{e, f\}$ the potential breakpoint at which the two weights become equal. We then compute $Z(\lambda)$ by sweeping through the sequence of potential breakpoints, maintaining the minimum spanning tree as we do. A potential breakpoint corresponds to an actual change in the MST exactly when the edge that is heavier before the breakpoint (say e) is not already in the MST and induces a cycle in the MST containing f; this can be tested in $O(\log n)$ time per potential breakpoint using the dynamic tree data structure of Sleator and Tarjan [SlTa83].

In this $O(m^2 \log n)$ time algorithm, there are many more potential than actual breakpoints, so it seems we are wasting work testing them all. Our idea is to apply sparsification to reduce the number of breakpoints we test.

We will need some notation. Let u be any node of a sparsification tree T_G of G. Then, $Z_u(\lambda)$ will denote the minimum spanning forest function[4] for G_u, b_u will denote the number of breakpoints of Z_u, and n_u and m_u will denote, respectively, the number of vertices and edges of G_u; $F_u(\lambda)$ will denote the minimum spanning forest of G_u at λ.

We will describe our algorithm in geometric terms, following Eppstein [Epp95]. Graph the weight functions of the individual edges in G_u as lines $w = w_e(\lambda)$, drawn in a plane with coordinates (λ, w). Then the potential breakpoints discussed above occur at the vertices of this arrangement of lines. We form a line segment arrangement A_u by taking subsegments of the lines; specifically, we include those portions of the line $w_e(\lambda)$ corresponding to values of λ for which e belongs to $F_u(\lambda)$. The following fact was proved by Eppstein [Epp95].

Lemma 2. *The arrangement A_u consists of $b_u + 1$ line segments that can be grouped to form $n_u - 1$ convex chains in the (λ, w) plane.*

The proof of this lemma is given by an argument in which we assign tokens to the edges in $F_u(\lambda)$; when two edges $\{e, f\}$ are swapped we pass the token from e to f; then the convex polygons described by the lemma are traced out by the $n_u - 1$ tokens.

[4] Since G_u need not be connected, we must consider spanning forests rather than spanning trees, but this does not lead to any additional complication.

Lemma 3. *Let p, q, r, and s be the four children of node u in the sparsification tree, and let $\{e, f\}$ be a swap occurring at $F_u(\lambda^*)$. Then there is a pair of line segments $\{s_e, s_f\}$ in the line segment arrangement $A_p \cup A_q \cup A_r \cup A_s$, on the two lines $w_e(\lambda)$ and $w_f(\lambda)$ respectively, that either cross at a point $(\lambda^*, w_e(\lambda^*))$ or both have endpoints at that point.*

Proof. We have already seen that the lines $w_e(\lambda)$ and $w_f(\lambda)$ cross at this point. But if e is in $F_u(\lambda^* - \epsilon)$, it is also in one of F_p, F_q, F_r, or F_s, so there is a line segment in the arrangement that goes through the points $(\lambda^* - \epsilon, w_e(\lambda^* - \epsilon))$. A symmetric argument shows that there is also a line segment corresponding to f through the points $(\lambda^* + \epsilon, w_f(\lambda^* + \epsilon))$. If one of these segments does not cross at $(\lambda^*, w_e(\lambda^*))$, it must be because the same two edges took part in a swap at a lower level of the sparsification tree, in which case both segments terminate at that point.

Our algorithm then is to calculate $Z_u(\lambda)$, and $A_u(\lambda)$ by traversing T_G in postorder, computing these quantities at each node by combining the same information from the node's children. Specifically, we compute the line segment intersections in the arrangement $A_p \cup A_q \cup A_r \cup A_s$, sort them by their λ-coordinates, and use these as potential breakpoints in an algorithm for computing Z_u similar to the folklore one described above: we sweep through the sequence of potential breakpoints, maintaining the minimum spanning tree as we do. A potential breakpoint corresponds to an actual change in the MST exactly when the edge that is heavier before the breakpoint (say e) is not already in the MST and induces a cycle in the MST containing f; this can be tested in $O(\log n)$ time per potential breakpoint using the dynamic tree data structure of Sleator and Tarjan [SlTa83].

Lemma 4. *The number of potential breakpoints tested by the algorithm above, at node u of the sparsification tree, is $O(m_u n_u)$.*

Proof. By Lemma 3 each potential breakpoint $\{e, f\}$ occurs at a point $(\lambda, w_e(\lambda))$ where a segment on line $w_e(\lambda)$ crosses another segment in one of A_p, A_q, A_r, or A_s. In particular, it corresponds to a point where line $w_e(\lambda)$ crosses one of the at most $4(n_u - 1)$ polygons into which (by Lemma 2) these four arrangements can be grouped. Each line can only cross each polygon at most twice, so there are at most $8m_u(n_u - 1)$ crossings total.

Theorem 5. *The algorithm described above computes $Z(\lambda)$ in $O(mn \log n)$ total time.*

Proof. It takes time $O(b_u \log b_u + m_u n_u)$ to compute the set of potential crossings [ChEd92], $O(m_u n_u \log n)$ to sort them by λ, and $O(m_u n_u \log n)$ to use the algorithm of Sleator and Tarjan to test each potential breakpoint. Thus the total time per node u is $O(m_u n_u \log n)$. The overall bound comes from summing this quantity over the nodes of the sparsification tree.

Remark. By using the bottom-up scheme just described, but replacing the arrangement-based approach by a suitable modification of the Eisner-Severance method, $Z(\lambda)$ can be generated in $O(T_{MST}(2n, n) \cdot b(m, n))$ time; this construction will be described in the full version of the paper. Depending on b and T_{MST}, this running time will be no worse than $O(n^{3/2} m \log\log^* n)$ and no better than $O(nm\alpha(n))$.

4 Parametric search problems

We now give an algorithm for problems (P1)–(P3) of the Introduction. For concreteness, we will describe our approach in the context of problem (P2), which asks one to find the value λ^* such that $Z(\lambda^*) = 0$; a similar approach works for the other problems. Problem (P2) arises in the solution of the minimum ratio spanning tree problem, a problem that is defined as follows. Given a graph where every edge e has two weights, a_e and b_e, find a spanning tree T of G such that the ratio $\sum_{e \in T} a_e / \sum_{e \in T} b_e$ is minimized [Cha77] (the b_e's are assumed to be either all negative or all positive). MRST arises in the design of communication networks, where the number a_e represents the cost of building link e, and b_e represents the time required to build that link. The goal is to find a tree that minimizes the ratio of total cost over construction time. Other applications of MRST are given elsewhere [CMV89, Meg83]. MRST can be reduced to a problem of type (P2) by associating with each edge $e \in G$ a linear weight function $w_e(\lambda) = a_e - \lambda b_e$ and letting $Z(\lambda)$ denote the weight of the minimum spanning tree relative to the weights $w_e(\lambda)$. It can be shown [Cha77] that the minimum ratio is the root λ^* of $Z(\lambda)$.

Our algorithm resembles the construction algorithm of the previous section in that it traverses the sparsification tree T_G from the bottom up, constructing Z_u for every node u; there are, however, some key differences. First, while the construction algorithm only required postorder traversal of the nodes, the search algorithm processes nodes level by level. Second, when processing a node u, the search algorithm computes $Z_u(\lambda)$ only for a restricted interval $\mathcal{I}$ containing λ^* within which Z_u has few breakpoints.

In more detail, the search algorithm is as follows. The initialization consists of (i) constructing the sparsification tree T_G of G up to a depth D to be specified later, (ii) initializing the search interval $\mathcal{I}$ to $(-\infty, +\infty)$, and (iii) generating Z_u within $\mathcal{I}$ for every node u at depth D in T_G. Next, for every value of k from $D-1$ down to 0, it does the following two steps. First, it narrows $\mathcal{I}$, so that, in addition to $\lambda^* \in \mathcal{I}$, the following property holds:

$$\sum_{u \in L_k} b_{\mathcal{I}}(Z_u) \le 2^{2k}, \tag{1}$$

where $b_{\mathcal{I}}(f)$ denotes the number of breakpoints of a piecewise-linear function within $\mathcal{I}$ and L_k denotes the set of nodes at depth k in T_G. The second step is to construct a representation of Z_u within $\mathcal{I}$ for every depth-k node u. This two-step processing of all nodes at depth k will be called *phase k*.

After phase 0 is complete, the root of T_G will have a complete description of $Z(\lambda)$ within $\mathcal{I}$. In the final step, the algorithm searches $Z(\lambda)$ to locate the value λ^* such that $Z_r(\lambda^*) = 0$

Next, we will describe the implementation of the main steps of the algorithm.

Constructing the Z_u's. For $u \in L_D$, Z_u is constructed using the algorithm of section 3. Nodes at depth $k < D$ are processed using a sparsification-based variant of the Eisner-Severance method [EiSe76]. The Eisner-Severance approach has the property that, for any node u of T_G, it generates a complete description of $Z_u(\lambda)$ within an interval $\mathcal{I}$ in time proportional to that required to do $b_{\mathcal{I}}(Z_u)$ evaluations of Z_u. Sparsification can be used to accelerate the evaluation of Z_u, assuming we have a suitable representation of Z_v for every child v of u. Such a representation must enable us to quickly retrieve two pieces of information for any $\lambda \in \mathcal{I}$: the value of $Z_v(\lambda)$ and the minimum spanning forest $F_u(\lambda)$ of G_u at λ. The representation can be implemented using balanced binary search trees [CLR90, FeSl94], in order to support retrieval in time logarithmic in $b_{\mathcal{I}}(Z_v)$; based on the known bounds on the number of breakpoints, the retrieval time will be $O(\log n_v)$.

Given representations of Z_p, Z_q, Z_r, Z_s for each of the children p, q, r, s of u, $Z_u(\lambda)$ can be evaluated in $O(T_{MST}(2n_u, n_u))$ time, since the minimum spanning forest of G_u is the same as the minimum spanning forest of the graph induced by $F_p(\lambda) \cup F_q(\lambda) \cup F_r(\lambda) \cup F_s(\lambda)$, which has at most $2n_u$ edges and n_u vertices. Our algorithm is then to use the Eisner-Severance method for every $u \in L_k$, but to rely on the preceding idea to ensure that the required evaluations are done on sparse graphs.

Lemma 6. *The method described above generates Z_u within $\mathcal{I}$ for every node u at depth k in total time $O(nm/2^D \log(n/2^D))$ for $k = D$ and in total time $O(n2^k \log\log^*(n/2^k))$ for $0 \leq k \leq D-1$.*

Proof. By Theorem 5, the total time to process all nodes at level D is

$$O\left(\sum_{u \in L_D} n_u m_u \log n_u\right) = O\left((n/2^D)\log(n/2^D) \cdot \sum_{u \in L_D} m_u\right)$$
$$= O\left((nm/2^D)\log(n/2^D)\right),$$

where the first equality follows from Lemma 1.

For each of the remaining levels, the time is proportional to that required to do $O(\sum_{u \in L_k} b_{\mathcal{I}}(Z_u))$ evaluations of Z_u on a graph with at most $n/2^{k-1}$ edges and $n/2^k$ vertices. Assuming (1), and taking $T_{MST}(m,n) = O(m \log \beta(m,n))$, this is $O(n2^k \log\log^*(n/2^k))$.

Narrowing the search interval. Narrowing requires the use of an *oracle*, a procedure that can determine the position of any λ-value λ_0 relative to λ^*; we refer to this as *resolving* λ_0. We will only describe the oracles for (P2); the other two problems can be handled analogously. Observe that any value $\lambda_0 \notin \mathcal{I}$ can be resolved in $O(1)$ time by determining its position relative to the endpoints of $\mathcal{I}$. Suppose, instead, that $\lambda_0 \in \mathcal{I}$. Assume that the slopes of all the edge costs are nonpositive (the case where they are all nonnegative is handled analogously). Then, $Z(\lambda)$ is a nonincreasing function and we have three possibilities [Meg83]: If $Z(\lambda_0) = 0$, then, $\lambda_0 = \lambda^*$. If $Z(\lambda_0) > 0$, then, $\lambda_0 < \lambda^*$. Finally, if $Z(\lambda_0) < 0$, then, $\lambda_0 > \lambda^*$. Thus, resolving a λ-value λ_0 reduces in $O(1)$ time to evaluating $Z(\lambda_0)$. We shall therefore focus on the evaluation problem.

Each phase of the search algorithm uses a different oracle; the oracle for phase k is denoted by $\mathcal{C}_k$. Oracle $\mathcal{C}_D$ evaluates Z using a minimum spanning tree algorithm with running time $T_{MST}(m, n)$. For $k < D$, oracle $\mathcal{C}_k$ computes $Z(\lambda)$ by retrieving the values of the $Z_u(\lambda)$'s for all depth $k+1$ nodes and then processing the nodes of the sparsification tree from depth $k+1$ up.

Lemma 7. *The running time of $\mathcal{C}_k$ is*

$$t_k = \begin{cases} T_{MST}(m, n) & \text{for } k = D \\ O\left(n 2^k \log\log^* \left(n/2^k\right)\right) & \text{for } 0 \le k \le D-1 \end{cases}$$

Proof. We only need to bound the running time of $\mathcal{C}_k$ for $k \le D-1$; this is

$$\begin{aligned} O\left(\sum_{j=k+1}^{0} \sum_{u \in L_j} T_{MST}(m_u, n_u)\right) &= O\left(\sum_{j=k+1}^{0} n 2^j \log \beta \left(n/2^{j-1}, n/2^j\right)\right) \\ &= O\left(n 2^k \log\log^* \left(n/2^k\right)\right). \end{aligned}$$

Note that we are using $T_{MST}(m, n) = O(m \log \beta(m, n))$.

We now describe how to narrow $\mathcal{I}$ using the oracles. Assume that at the beginning of phase k, $\sum_{u \in L_{k+1}} b_{\mathcal{I}}(Z_u) \le 2^{2k+2}$; our objective is to find a subinterval of $\mathcal{I}$ such that (1) holds. To ensure that our assumption holds at the beginning of phase $D-1$, it will be necessary to do some narrowing immediately after the initialization step, in which the nodes at depth D are processed. By Lemma 1 and Gusfield's bound on $b(m, n)$ [Gus80], for $\mathcal{I} = (-\infty, +\infty)$, we have

$$\sum_{u \in L_D} b_{\mathcal{I}}(Z_u) = O\left(\sum_{u \in L_D} m_u \sqrt{n_u}\right) = O\left(\sqrt{n} m / 2^{D/2}\right).$$

One can narrow $\mathcal{I}$ so that $\sum_{u \in L_D} b_{\mathcal{I}}(Z_u) \le 2^{2D}$ by making $O(\log(m\sqrt{n}/2^{5D/2}))$ calls to $\mathcal{C}_D$. The approach is standard: do a series of steps each of which halves the number of breakpoints by locating the median breakpoint within $\mathcal{I}$, and, by resolving it, decides whether to discard the part of $\mathcal{I}$ before the median, or the part after the median [FeSl95].

To understand how narrowing works, it helps to examine the relationship between breakpoints of functions associated with depth-k nodes, and those associated with nodes at depth $k+1$. Following notation similar to that of section 3, A_v will denote the arrangement of line segments associated edges appearing in the minimum spanning tree of G_v for $\lambda \in \mathcal{I}$; i.e., we include those portions of the line $w_e(\lambda)$ corresponding to values of $\lambda \in \mathcal{I}$ for which e belongs to $F_v(\lambda)$. We have the following complement to Lemma 3, which is similar to a fact proved in an earlier paper [FeSl95].

Lemma 8. *Let p, q, r, and s be the four children of node u in the sparsification tree. Then, there is a breakpoint of Z_u at $\lambda_0 \in \mathcal{I}$ only if there is a pair of line segments $\{s_e, s_f\}$ in the line segment arrangement $A_p \cup A_q \cup A_r \cup A_s$, on the two cost lines $w_e(\lambda)$ and $w_f(\lambda)$ respectively, that either cross at a point $(\lambda_0, w_e(\lambda_0))$ or both have endpoints at that point.*

Thus, when narrowing the search interval, we can focus on breakpoints of the Z_v's associated with nodes at depth $k+1$, and on the intersections between line segments associated with siblings at depth $k+1$, in a manner similar to the approach in Section 3. Unlike what was done there, however, we cannot simply generate all intersection points, since this would be too time-consuming. We shall, instead, repeatedly use a procedure Select, which allows us to select specific intersection points in an arrangement of cost lines.

Select$(A, k, \mathcal{I})$: Given an arrangement of lines A, an integer k, and an interval $\mathcal{I}$, return the intersection point in A with the kth largest abscissa amongst those intersection points whose abscissa falls within $\mathcal{I}$.

Agarwal [Aga91] showed that Select can be implemented in $O(r \log r)$ time, where r is the number of lines in the arrangement.

Roughly speaking, the interval narrowing algorithm uses Select to generate sample points in the middle of each subinterval of $\mathcal{I}_u$; the oracle will then be used to halve these intervals. This process is repeated logarithmically-many times until the interval is small enough. The details of the algorithm are given below. In the description, we shall write l_e to denote the line on the (λ, w) plane traced by the cost of $e \in E(G)$, and we shall write $\mathcal{I}(B)$ to denote the subdivision of $\mathcal{I}$ induced by a set of λ-values B. The algorithm assumes that, prior to being called, $\lambda^* \in \mathcal{I}$ and $\sum_{u \in L_{k+1}} b_{\mathcal{I}}(Z_u) \leq 2^{2k+2}$.

Narrow$(k,\mathcal{I})$
$N := \binom{n/2^k}{2}$
for each $u \in L_k$ **do**
  $B_u := \{\lambda \in \mathcal{I} : \lambda$ is a breakpoint of Z_v for some child v of $u\}$
for $i := 2$ **to** $\log N$ **do**
  for each $u \in L_k$ **and each** subinterval $\mathcal{J}$ of $\mathcal{I}(B_u)$ **do**
    $C_u := \{l_e : e \in F_v(\lambda), \lambda \in \mathcal{J},$ for some child v of $u\}$
    Comment: By definition of B_u, $F_v(\lambda)$ is the same for all $\lambda \in \mathcal{J}$
    Add to B_u the λ-coordinate of Select$(C_u, N/2^i, \mathcal{J})$
  Reduce $\mathcal{I}$ so that $\lambda^* \in \mathcal{I}$ and $|L \cap \mathcal{I}| \leq 2^{2k}$, where $L = \bigcup_{u \in L_k} B_u$
return $\mathcal{I}$

Lemma 9. Narrow *computes an interval $\mathcal{I}$ containing λ^* satisfying (1) in $O(n2^k \log^2(n/2^k))$ time.*

Proof. Suppose that, at the beginning of each iteration of the loop beginning at line 3, $|\bigcup B_u| \le 2^{2k+2}$; by assumption this is true at the beginning of the first iteration. Thus, steps 4–6 take $O(n2^k \log(n/2^k))$ time and, in step 7, we will have $|L| \le 2^{2k+3}$.

Step 7 can be implemented with three calls to $\mathcal{C}_k$ as follows. First, discard all values in L that fall outside $\mathcal{I}$. Next, in $O(|L|)$ time [CLR90], compute the median value λ_m of L, and apply $\mathcal{C}_k$ to resolve its position relative to λ^*. The answer allows us to either discard the portion of $\mathcal{I}$ to the left of λ_m or the portion to the right of λ_m, thereby halving the number of elements of L in $\mathcal{I}$. Repeating this process at most two additional times will ensure that $|L \cap \mathcal{I}'| \le 2^{2k}$. Hence, the total time for Step 7, including the median computations, is $O(2^{2k} + t_k)$. Since statements 3–7 are iterated $O(\log(n/2^k))$ times, the running time of the narrowing algorithm is $O(n2^k \log^2(n/2^k) + t_k \log(n/2^k))$; the time bound follows from Lemma 7.

After the ith iteration of statements 3–7, each subinterval of $\mathcal{I}(B_u)$ will have at most $\binom{n/2^k}{2}/2^i$ intersection points of C_u. Thus, after the last iteration, for every $u \in L_k$, no subinterval of $\mathcal{I}(B_u)$ will have any such intersection points. The total size of the B_u's will be $O(2^{2k})$; this, together with Lemma 8, implies that the interval returned by NARROW satisfies (1).

Analysis. We now state the main result of this section:

Theorem 10. *The search algorithm runs in $O(n\sqrt{m}\log^2(n^2/m))$ time.*

Proof. We choose $D = \lceil 0.5 \log m \rceil$. Thus, the initialization (including the processing of all nodes at depth D) takes $O(n\sqrt{m}\log(n^2/m))$. The initial narrowing prior to the beginning of phase $D-1$ takes $O(m \log\beta(m,n)\log(n^2/m)$ time. By Lemmas 6 and 9, the total running time of phases $D-1$ to 0 is

$$O\left(n \sum_{k=D}^{0} 2^k \log^2(n/2^k)\right) = O\left((n2^D)\log^2\left(n/2^D\right)\right)$$
$$= O\left(n\sqrt{m}\log^2\left(n^2/m\right)\right),$$

At the end of the last phase, we will have an representation of $Z(\lambda)$ within an interval $\lambda^* \in \mathcal{I}$. Since the number of breakpoints of Z is polynomially-bounded in n, a binary search on its representation will enable us to locate λ^* in $O(\log n)$ times. Thus, the total running time is $O(n\sqrt{m}\log^2(n^2/m))$.

5 Discussion

We have presented algorithms for construction and search problems arising from parametric minimum spanning trees. While our time bounds for constructing Z improve on the Eisner-Severance approach, the latter seems more output-sensitive. It is an open problem to determine if there is a faster, perhaps output-sensitive, algorithm for the construction problem. Finally, we note that our search algorithm is inefficient for non-dense graphs; whether or not it is possible to find an algorithm whose running time matches T_{MST} remains open.

Acknowledgements

The first author thanks Jens Lagergren for helpful discussions and Vladimir Estivill-Castro, who hosted him during a visit to LANIA, in Xalapa, Mexico, where portions of this paper were written.

References

[Aga91] P.K. Agarwal. *Intersection an Decomposition Algorithms for Planar Arrangements.* Cambridge University Press, Cambridge, 1991.

[CEGS92] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair, and parametric searching. In *Proceedings of the 8th Annual ACM Symposium on Computational Geometry,* pp. 120– 129 (1992).

[CMV89] P.M. Camerini, F. Maffioli, and C. Vercellis. Multi-constrained matroidal knapsack problems. *Mathematical Programming* 45:211–231, 1989.

[CoMe93] E. Cohen and N. Megiddo. Maximizing concave functions in fixed dimension. In *Complexity in Numerical Computations*, P.M. Pardalos, ed., pp. 74–87, World Scientific Press 1993.

[Cole87] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. Assoc. Comput. Mach.*, 34(1):200–208, 1987.

[Cha77] R. Chandrasekaran. Minimal ratio spanning trees. *Networks*, 7: 335–342, 1977.

[ChEd92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. Assoc. Comput. Mach.*, 39:1–54, 1992.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[EiSe76] M.J. Eisner and D.G. Severance. Mathematical techniques for efficient record segmentation in large shared databases. *J. Assoc. Comput. Mach.*, 23:619–635, 1976.

[EGIN92] D. Eppstein, Z. Galil, G.F. Italiano, and A. Nissenzweig. Sparsification — a technique for speeding up dynamic graph algorithms. In *Proc. 33rd Annual Symposium on Foundations of Computer Science*, pp. 60–69, 1992.

[EGI93] D. Eppstein, Z. Galil, and G.F. Italiano. Improved sparsification. Tech Report 93-20, Department of Computer Science, University of Califonia, Irvine, April, 1993.

[Epp95] D. Eppstein. Geometric lower bounds for parametric matroid optimization. Tech Report 95-11, Department of Computer Science, University of Califonia, Irvine, March, 1995. Preliminary version in *27th Annual Symp. on Theory of Computing*, 1995.

[FeSl94] D. Fernández-Baca and G. Slutzki. Optimal parametric search on graphs of bounded tree-width. In *Proc. 4th Scandinavian Workshop on Algorithm Theory*, pp. 155–166, LNCS 824, Springer-Verlag, 1994. To appear in *J. Algorithms.*

[FeSl95] D. Fernández-Baca and G. Slutzki. Linear-time algorithms for parametric minimum spanning tree problems on planar graphs. In *Proc. Latin American Conference on Theoretical Informatics*, pp. 257–271, LNCS 911, Springer-Verlag, 1995.

[Fre85a] G.N. Frederickson. Data structures for on-line updating of minimum spanning trees. *SIAM J. Comput.* 14:781–798, 1985.

[Fre90] G.N. Frederickson. Optimal algorithms for partitioning trees and locating p-centers in trees. Technical Report CSD-TR 1029, Department of Computer Science, Purdue University, October 1990.

[Fre91] G.N. Frederickson. Ambivalent data structures for dynamic 2-edge connectivity and k-smallest spanning trees. In *Proc. 32nd Annual Symp. on Foundations of Computer Science*, pp. 632–641, 1991.

[FrWi90] M. Fredman and D. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In *Proc. 31st Annual IEEE Symp. on Foundations of Computer Science*, 1990, pp. 719–725.

[GGST86] H.N. Gabow, Z. Galil, T. Spencer, and R.E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.

[Gus80] D. Gusfield. *Sensitivity analysis for combinatorial optimization.* Technical Report UCB/ERL M80/22, University of California, Berkeley, May 1980.

[Gus83] D. Gusfield. Parametric combinatorial computing and a problem in program module allocation. *J. Assoc. Comput. Mach.*, 30(3):551–563, 1983.

[HaTa89] R. Hassin and A. Tamir. Maximizing classes of two-parametric objectives over matroids. *Math. Oper. Res.*, 14:362-375, 1989.

[ISN81] H. Ishii, S. Shiode, and T. Nishida. Stochastic spanning tree problem. *Discrete Applied Mathematics*, 3:263–273, 1981.

[KaIb83] N. Katoh and T. Ibaraki. On the total number of pivots required for certain parametric combinatorial optimization problems. Technical Report Working Paper 71, Inst. Econ. Res., Kobe Univ. Commerce, 1983.

[KKT94] D.R. Karger, P.N. Klein, and R.E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. *Manuscript.* A preliminary version appeared in *STOC 94*.

[MaSc93] J. Matoušek and O. Schwartzkopf. A deterministic algorithm for the three-dimensional diameter problem. In *Proceedings of 25th Annual Symposium on Theory of Computing*, pp. 478–484 (1993).

[Meg79] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4:414–424, 1979.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach.*, 30(4):852–865, 1983.

[ShTo94] M. Sharir and S. Toledo. Extremal polygon containment problems. *Computational Geometry*, 4:99–118, 1994.

[SlTa83] D.D.K. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[Tol93a] S. Toledo. Maximizing non-linear convex functions in fixed dimension. In *Complexity in Numerical Computations*, P.M. Pardalos, ed., pp. 74–87, World Scientific Press 1993. A preliminary version appeared in FOCS 92.