Parametric and Kinetic Minimum Spanning Trees

Pankaj K. Agarwal * David Eppstein † Leonidas J. Guibas ‡ Monika R. Henzinger

Abstract

We consider the parametric minimum spanning tree problem, in which we are given a graph with edge weights that are linear functions of a parameter λ and wish to compute the sequence of minimum spanning trees generated as λ varies. We also consider the kinetic minimum spanning tree problem, in which λ represents time and the graph is subject in addition to changes such as edge insertions, deletions, and modifications of the weight functions as time progresses. We solve both problems in time $O(n^{2/3}\log^{4/3} n)$ per combinatorial change in the tree (or randomized $O(n^{2/3}\log n)$ per change). Our time bounds reduce to $O(n^{1/2}\log^{3/2} n)$ per change ($O(n^{1/2}\log n)$ randomized) for planar graphs or other minor-closed families of graphs, and $O(n^{1/4}\log^{3/2} n)$ per change ($O(n^{1/4}\log n)$ randomized) for planar graphs with weight changes but no insertions or deletions. [1]

*Center for Geometric Computing, Department of Computer Science, Duke Univ., Durham, NC, 27708-0129; http://www.cs.duke.edu/∼pankaj/; pankaj@cs.duke.edu. Work supported in part by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award, and by a grant from the U.S.-Israeli Binational Science Foundation.

†Dept. of Information and Computer Science, Univ. of California, Irvine, CA 92697-3425; http://www.ics.uci.edu/∼eppstein/; eppstein@ics.uci.edu. Work supported in part by NSF grant CCR-9258355 and by matching funds from Xerox Corp.

‡Dept. of Computer Science, Stanford Univ., Stanford, CA, 94305; http://graphics.stanford.edu/∼guibas/; guibas@cs.stanford.edu. Work supported in part by Army Research Office MURI grant DAAH04-96-1-0007 and NSF grant CCR-9623851

Compaq Systems Research Center, 130 Lytton Ave., Palo Alto, CA, 94301-1044; http://www.research.digital.com/SRC/personal/Monika_Henzinger/home.html; monika@pa.dec.com.

1. Introduction

The *parametric minimum spanning tree problem* deals with minimum spanning trees of weighted graphs, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which the weight of each edge is a linear function of some parameter λ, instead of a real number. That is, the weight of each edge $e \in \mathcal{E}$, $w_e(\lambda)$, is of the form $x_e - \lambda y_e$, where x_e, y_e are real numbers. As λ varies, the weight of each edge varies and therefore the weight of the minimum spanning tree of $\mathcal{G}$ also varies. At certain discrete values of λ, the minimum spanning tree itself changes. The two central questions related to this problem are: how many different trees does $\mathcal{G}$ have as λ varies from $-\infty$ to $+\infty$, and how can one compute this sequence of trees?

As Katoh [23, 24] has described, parametric minimum spanning trees can be used to solve many other problems in combinatorial optimization, in which one seeks to find a tree T minimizing a function of the form $f(X, Y)$ where f is quasiconcave and $X = \sum_{e\in T} x_e$ and $Y = \sum_{e\in T} y_e$. For example, the problem of computing a spanning tree that minimizes the ratio of cost to reliability can be expressed in the above form by setting $f(X, Y) = X \exp(Y)$, where x_e is the cost of edge e, $y_e = -\ln(1 - p_e)$, with p_e the failure probability of edge e. The stochastic programming problem of finding a tree with high probability of having low weight can be expressed with the choice $f(X, Y) = X + \sqrt{Y}$, and the problem of finding a tree with the minimum variance in edge weight can be similarly expressed with $f(X, Y) = X - Y^2$. For each of these problems, any spanning tree of $\mathcal{G}$ gives rise to a point (X, Y) in the plane. The quasiconcave nature of f implies that the optimum tree is a vertex of the convex hull of the set of all such points, and the trees giving rise to the vertices of the convex hull are exactly those which are enumerated in the parametric minimum spanning tree problem for $\mathcal{G}$ with edge weights $x_i - \lambda y_i$. Thus, one can solve the original static optimization problem by finding the best of the trees listed by a parametric minimum spanning tree algorithm. Note especially that in this type of application, there is no need to output each tree explicitly; we can spend sublinear time per tree as long as we can determine fast the two numbers X and Y for each tree.

Parametric optimization problems such as this are a special case of a more general class of algorithms, *kinetic algorithms*, recently proposed by Guibas et al. [3, 19]. A *kinetic* problem focuses on exploiting the coherence implicit in the continuous evolution of a system whose elements move or change according to known laws, while allowing arbitrary changes in these motion laws in an on-line fashion. A general kinetic problem combines dynamic data structures (in which sets of objects undergo single-object insertions and deletions) with parametric optimization (in which static objects have continuously varying weights). In such a problem, one starts with a parametric problem in which the parameter λ represents time; as time progresses, objects may be deleted or inserted, or their weight functions changed, and the task is to maintain the optimal solution at each value of time. Kinetic algorithms model well real-world phenomena where objects move along trajectories that are predictable in the short term, but subject to unpredictable changes in the long term.

We distinguish two possible types of kinetic algorithms for the minimum spanning tree problem:

- A *structurally kinetic* algorithm can handle arbitrary insertions or deletions of parametrically weighted edges. Edge weight function changes can be simulated by deleting and re-inserting the edge.

- A *functionally kinetic* algorithm can only handle updates that change the weight function of an existing edge. Edge deletions and insertions can be simulated by changing weights to or from some very large value; however this simulation comes at the cost of increasing the number of edges (and perhaps violating restrictions such as that the graph remain planar).

In this paper, we provide efficient algorithms for both the functionally kinetic and structurally kinetic minimum spanning tree problems. Since the parametric minimum spanning tree problem is a special case of either type of kinetic problem, our algorithms also apply *a fortiori* in the parametric case.

Parametric and kinetic minimum spanning trees form an interesting combination of graph theory and computational geometry: the minimum spanning tree part of the problem is purely graph-theoretic, while the weight functions can be viewed as lines in a weight-time plane, lending the problem a geometric flavor. Our solution technique, too, combines graph algorithms and computational geometry: we use sparsification and clustering techniques common to many dynamic graph algorithms, with convex hull data structures representing sets of edges in each cluster. We also apply several other techniques including parametric search (a technique of Megiddo [26] for turning decision algorithms into optimization algorithms, commonly used in both parametric optimization and computational geometry; see, e.g., [1]) and a data structure for maintaining convex hulls of a point set subject to insertions and undo operations.

1.1. Notation

Throughout, we assume that we have a weighted, connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n vertices and m edges, in which the weight of each edge e is a linear function $w_e(\lambda) = x_e - \lambda y_e$. (For structurally kinetic problems, n and m denote the number of vertices and edges at some particular point in the course of the algorithm.) The graph may have multiple adjacencies. Let $MST_{\mathcal{G}}(\lambda)$ denote a minimum spanning tree of $\mathcal{G}$ for the weights $w_e(\lambda)$; if we break ties in favor of earlier-numbered edges, the $MST_{\mathcal{G}}(\lambda)$ is well defined and piecewise constant. If the underlying graph is obvious from the context, we will simply use $MST(\lambda)$ to denote $MST_{\mathcal{G}}(\lambda)$. Let $\mathcal{T} = \mathcal{T}(\mathcal{G})$ be the set of all minimum spanning trees of $\mathcal{G}$ as the value of λ varies from $-\infty$ to $+\infty$. Set $k = k(\mathcal{G}) = |\mathcal{T}(\mathcal{G})|$.

The goal of a parametric minimum spanning tree algorithm is to compute the set $\mathcal{T}$. We denote by $P(n, m)$ the maximum value of $k(\mathcal{G})$, where the maximum is taken over all (parametric) graphs with n vertices and m edges. Dey [7] recently proved that $P(n, m) = O(mn^{1/3})$; the best lower bound known is $P(n, m) = \Omega(m\alpha(n))$ [9].

In a kinetic minimum spanning tree algorithm, the parameter λ represents time, and the goal is to maintain $MST(\lambda)$ dynamically as the graph undergoes changes either to its structure or to its weight functions. We denote by $K(n, m)$ the maximum number of distinct trees formed by a kinetic problem on a graph with n vertices subject to m edge insertions, edge deletions, or weight change operations starting from an empty graph. A third parameter for the number of edge weight changes is superfluous, since the number of distinct trees in a graph with $x > m$ changes is easily seen to be $\Theta((x/m)K(n, m))$.) We do not use separate notations for the number of trees in functionally and structurally kinetic problems, since the simulations between each kind of kinetic algorithm causes these numbers to be the same to within a constant factor.

If a sequence of structurally kinetic changes to a graph were known in advance, then we could simulate the kinetic problem by a parametric problem: simply replace each edge e by a three-edge path $e_1e_2e_3$, with the weight of e_1 equal to that of the kinetic edge, with e_2 having a highly negatively sloped weight function, and with e_3 having a highly positively sloped weight function, chosen so that the interval in which e_2 is the largest of the three weights is exactly the lifetime of e. Then, in this modified graph, the minimum spanning tree will consist of the two least weight edges in each three-edge path, together with a third edge in a subset of paths corresponding exactly to the minimum spanning tree of the original graph. (Essentially the same three-edge-path construction was used in the lower bound of [9].) For this reason $K(n,m) \le P(n+2m, 3m) = O(m^{4/3})$.

We will also consider the special cases of planar and minor-closed graph families, for which the number of distinct trees may be smaller than in the case of general graphs. For any minor-closed family $\mathcal{F}$, we let $P_{\mathcal{F}}(n)$ denote the maximum number of distinct trees in a parametric problem on an n-vertex graph, and $K_{\mathcal{F}}(n)$ denote the maximum number of distinct trees in a structurally kinetic problem. Since minor-closed graph families consist only of sparse graphs, there is no need to include a second parameter m. The method above of simulating edges by paths shows that the maximum number of distinct trees in a functionally kinetic problem is $\Theta(P_{\mathcal{F}}(n))$ whenever $\mathcal{F}$ is closed under replacement of an edge by a series-parallel graph (in particular, this is true for planar graphs); however it does not seem possible to use this method to bound $K_{\mathcal{F}}(n)$ by $P_F(n)$.

1.2. History and New Results

Study of minimum spanning trees has a long rich history [18]. Currently, it is known how to compute the minimum spanning tree in randomized linear expected time [22] or deterministically in time $O(m\alpha(m,n)\log\alpha(m,n))$ [5]. Efficient algorithms have been developed for maintaining the minimum spanning tree of a graph as edges are inserted into or deleted from the graph [11, 15, 20, 21]. The parametric minimum spanning tree problem has also been previously studied, most recently by Fernández-Baca et al. [14]. In that paper an algorithm was described that takes time $O(mn\log n)$ to list all trees. However this still remains far from Dey's bound of $O(mn^{1/3})$ on the number of such trees. Parametric optimization problems have been studied for several other graph problems as well; see [13, 28] for a sample of such results. There are no known previous algorithms for the kinetic minimum spanning tree problem. A related problem, which has been studied, is the *kinetic Euclidean MST* problem [4], in which we want to list all different Euclidean minimum spanning trees of a set of points, each of which is moving along a line or curve.

Let p denote the number of edge insertions, edge deletions, or minimum spanning tree topology changes. Here we show the following results, substantially improving what was previously known.

- We can maintain a structurally kinetic graph, and keep track of the minimum spanning tree, in total time $O(\min\{pm^{2/3}\log^{4/3} m, K(n,m)n^{2/3}\log^{4/3} n\})$. If we allow randomization, the expected time is $O(\min\{pm^{2/3}\log m, K(n,m)n^{2/3}\log n\})$.

- For any structurally kinetic graph belonging to a minor-closed family, we can keep track of the minimum spanning tree in total time $O(pn^{1/2}\log^{3/2} n)$. If we allow randomization, the expected total time is $O(pn^{1/2}\log n)$.

- For any parametric or functionally kinetic graph belonging to a minor-closed family, we can keep track of the minimum spanning tree in total time $O(n^{3/2} + P(n)n^{1/4}\log^{3/2} n)$. If we allow randomization the expected total time is $O(n^{3/2} + P(n)n^{1/4}\log n)$. For planar graphs the $O(n^{3/2})$ term in these bounds can be removed. With Dey's bound $P(n) = O(n^{4/3})$, our total time is worst-case bounded by $O(n^{19/12}\log^{3/2} n)$, or randomized expected time $O(n^{19/12}\log n)$.

2. Sparsification

Sparsification [11] is a divide-and-conquer technique used in dynamic algorithms, whereby the edges of a graph are split recursively into subsets, a *certificate*[2] for each subset is maintained dynamically, and the overall property is maintained using a dynamic graph algorithm applied to the union of these certificates. Fernández-Baca et al. [14] showed that a similar idea applies also to parametric problems. We combine both of these applications of sparsification to obtain an efficient data structure for the kinetic problem, which is a common generalization of dynamic and parametric problems.

2.1. General Graph Sparsification

The key result needed to apply sparsification is the following lemma, which shows that graphs can be replaced by *sparse certificates* without changing the solution to the minimum spanning tree problem. This is a result about statically weighted graphs, but it holds *a fortiori* for any particular value of λ occurring in a kinetic algorithm.

Lemma 1 (Eppstein et. al [11])**.** *The minimum spanning tree of a graph $\mathcal{G} \cup \mathcal{H}$ is equal to the minimum spanning tree of the subgraph formed by the union of the minimum spanning trees of $\mathcal{G}$ and $\mathcal{H}$.*

We then use a divide and conquer approach, applying this lemma to combine solutions to subproblems. The parametric case of the following lemma is implicit in [14].

Lemma 2. *Suppose that we have a data structure that can solve structurally kinetic minimum spanning tree problems in time $O(f(m,n))$ per insertion, deletion, or topology change, and that $f(m,n) = \Omega(m^c)$ for some constant $c > 0$. Then we can solve structurally or functionally kinetic problems in time $O(K(n,m)f(2n,n))$, and parametric problems in time $O(P(n,m)f(2n,n))$.*

[2] The certificates are subgraphs of the original graph; they should not be confused with the distinct notion of certificate used in kinetic proofs [19].

Proof: We outline the general method here; see [14] for details. We divide the edges of the graph arbitrarily into two equal subsets, and solve recursively the kinetic problems for each subset. The solutions to these subproblems consist of a sequence of changes to the minimum spanning trees of the subproblems. We use our assumed data structure to solve a kinetic problem on the union of these two trees, by merging these two sequences of updates. At any time there are at most $2n$ edges in the kinetic problem, and the number of updates coming from each subset is $K(n, m/2)$, hence we get a recurrence of $T(m, n) = 2T(m/2, n) + O(K(n, m) f(2n, n))$ for the overall problem. As described here, this might lead to an additional logarithmic factor over the stated bounds; the "improved sparsification" technique [11] avoids this extra logarithm by reducing the number of vertices as well as the number of edges in the recursive subproblems. □

2.2. Separator-Based Sparsification Certificates

We applied sparsification to speed up our kinetic algorithms for general graphs. We now similarly apply *separator based sparsification* [12] to speed up our algorithms for planar graphs and for minor-closed graph families. The basic idea of this approach is to divide the graph into two subgraphs by a *separator*, a small set X of vertices shared by both subgraphs, so that each subgraph has only a constant fraction of the original graph's vertices. For planar graphs and other minor closed families, there always exist separators of size $O(n^{1/2})$, and a recursive decomposition into separators can be found in time $O(n^{3/2})$ [2]; the time complexity of this step can be improved to $O(n)$ for planar graphs [17]).

As in the general graph sparsification, we recursively solve a problem in each subgraph, and construct a *certificate* so that the overall MST can be found by combining the two subgraph certificates in a single kinetic problem. However, for this approach to work, the certificate must have size proportional to the number of vertices in the separator X, not to the size of the whole subgraph.

Suppose we are given two edge-disjoint subgraphs $\mathcal{G}$ and $\mathcal{H}$; their union is the whole graph, and their (vertex) intersection is a separator X. We describe how to find a certificate $C(\mathcal{G}, \mathcal{H})$ for $\mathcal{G}$. The certificate is a graph of size $O(|X|)$, obtained by contracting certain vertices and edges of $\mathcal{G}$. The construction for $\mathcal{H}$ is completely symmetric. Our construction works for static weights; we will describe later how to make it kinetic.

Without loss of generality (by inserting dummy edges of low weight, as described in more detail in a later section), we can assume that every vertex in $\mathcal{G}$ has degree at most three. Let T be a forest formed by taking a spanning forest of $\mathcal{H}$, removing all degree-one vertices not in X, and contracting all degree-two vertices not in X. Then T has at most $2|X| - 1$ edges, and $\mathcal{N}(\mathcal{G}) = \mathcal{G} \cup T$ is a minor of the overall graph. We form a parametric problem by keeping the weights of edges in $\mathcal{G}$ fixed and assigning a weight function $w_e(\lambda) = \lambda$ to the edges in T. Then in this parametric problem, $MST_{\mathcal{N}(\mathcal{G})}(+\infty) \cap \mathcal{G}$ is just the minimum spanning tree of $\mathcal{G}$ itself, as paths through T are too expensive to be useful for connecting nodes in G. Also, $MST_{\mathcal{N}(\mathcal{G})}(-\infty)$ is a tree formed by adding some of the the edges and vertices of T, while removing some edges from $MST_{\mathcal{N}(\mathcal{G})}(+\infty)$ (we now use all the cheap bypasses provided by T).

Lemma 3. *Any edge e in $MST_{\mathcal{N}(\mathcal{G})}(-\infty) \cap \mathcal{G}$ is an edge in the minimum spanning tree of $\mathcal{G} \cup \mathcal{H}$.*

Lemma 4. *Any edge e in $\mathcal{G}$ that is not in $MST_{\mathcal{N}(\mathcal{G})}(+\infty)$ is not in the minimum spanning tree of $\mathcal{G} \cup \mathcal{H}$.*

We call an edge e of $\mathcal{G}$ *uncertain* if $e \in MST_{\mathcal{N}(\mathcal{G})}(+\infty)$ but $e \notin MST_{\mathcal{N}(\mathcal{G})}(-\infty)$. We cannot be sure whether an uncertain edge is in the minimum spanning tree of the entire graph without knowing the weights of the edges in $\mathcal{H}$. If we delete from $MST_{\mathcal{N}(\mathcal{G})}(-\infty)$ all the vertices and edges not in $\mathcal{G}$, this tree is split into $|X|$ connected components, each containing exactly one vertex of X. It can be checked that the uncertain edges of $\mathcal{G}$ connect these components to form $MST(\mathcal{G})$. This immediately implies the following.

Lemma 5. *Any uncertain edge is part of a path in $MST_{\mathcal{N}(\mathcal{G})}(+\infty)$ between two vertices of X. The number of uncertain edges is $|X| - 1$.*

The certificate $C(\mathcal{G}, \mathcal{H})$ for $\mathcal{G}$ is constructed from $MST_{\mathcal{N}(\mathcal{G})}(+\infty) \cap \mathcal{G}$ as follows: First, we assign all uncertain edges their weights in $\mathcal{G}$, but the other edges are given a weight of $-\infty$. Then, we repeatedly remove from $MST_{\mathcal{N}(\mathcal{G})}(+\infty) \cap \mathcal{G}$ all degree-one vertices not in X. Finally, as long as the remaining tree contains a degree-two vertex that is not in X and that is adjacent to two edges with weight $-\infty$, we remove that vertex by contracting one of the adjacent edges.

Lemma 6. *The edges in the minimum spanning tree of $\mathcal{G} \cup \mathcal{H}$ are the disjoint union of three sets: the edges in $MST_{\mathcal{N}(\mathcal{G})}(-\infty) \cap \mathcal{G}$, the edges in $MST_{\mathcal{N}(\mathcal{H})}(-\infty) \cap \mathcal{H}$, and the edges in the minimum spanning tree of $C(\mathcal{G}, \mathcal{H}) \cup C(\mathcal{H}, \mathcal{G})$.*

Lemma 7. *If the weights of $\mathcal{G}$ change as part of a parametric or functionally kinetic problem, $C(\mathcal{G}, \mathcal{H})$ undergoes $O(P_{\mathcal{F}}(|\mathcal{V}|))$ structural changes.*

Proof: This follows from the fact that the structure of $C(\mathcal{G}, \mathcal{H})$ is determined by the structure of the two minimum spanning trees $MST_{\mathcal{N}(\mathcal{G})}(+\infty)$ and $MST_{\mathcal{N}(\mathcal{G})}(-\infty)$. Each change to one of these spanning trees causes $O(1)$ changes to $C(\mathcal{G}, \mathcal{H})$. □

2.3. Separator-Based Sparsification

Lemma 8. *Let $\mathcal{F}$ be a minor-closed graph family. Suppose that we have a data structure that can solve structurally kinetic minimum spanning tree problems restricted to graphs in $\mathcal{F}$ in time $O(f(n))$ per insertion, deletion, or topology change, where $f(n) \geq \log n$ and $P_{\mathcal{F}}(n)f(n) = \Omega(n^c)$ for some $c > 1$. Then we can solve functionally kinetic or parametric problems on graphs in $\mathcal{F}$ in time $O(n^{3/2} + P_{\mathcal{F}}(n)f(\sqrt{n}))$. If $\mathcal{F}$ contains only planar graphs, we can solve these problems in time $O(P_{\mathcal{F}}(n)f(\sqrt{n}))$.*

Proof: We form a separator decomposition of the graph. At each level of the decomposition, we have two subgraphs $\mathcal{G}$ and $\mathcal{H}$ the union of which is a subgraph at the next higher level. We maintain the four trees $MST_{\mathcal{N}(\mathcal{G})}(+\infty)$, $MST_{\mathcal{N}(\mathcal{H})}(+\infty)$, $MST_{\mathcal{N}(\mathcal{G})}(-\infty)$, and $MST_{\mathcal{N}(\mathcal{H})}(-\infty)$ described in the previous section, and the certificate $C(\mathcal{G},\mathcal{H})$ or $C(\mathcal{H},\mathcal{G})$ derived from those trees. Recall that the certificates are obtained by contracting subtrees of $MST_{N(\mathcal{G})}(+\infty)$ and $MST_{N(\mathcal{H})}(+\infty)$. In order to update the certificates efficiently, we maintain these contracted subtrees using the dynamic-tree data structure by Sleator and Tarjan [27]. The two trees $MST_{\mathcal{N}(\mathcal{G}\cup\mathcal{H})}(+\infty)$ and $MST_{\mathcal{N}(\mathcal{G}\cup\mathcal{H})}(-\infty)$ can then be found from this information together with the solution to two structurally kinetic problems on the graphs $C(\mathcal{G},\mathcal{H}) \cup C(\mathcal{H},\mathcal{G})$ and $C(\mathcal{G},\mathcal{H}) \cup C(\mathcal{H},\mathcal{G}) \cup T$ (where T is a contracted tree representing a subgraph at a higher level of the recursion, and has weights that do not vary).

The resulting system of data structures contains two kinetic problems at each level of the recursion, each of which undergoes a number of changes proportional to $P_{\mathcal{F}}(x)$ where x is the size of the subgraph. The overall time bound therefore satisfies the recurrence

$$T(n) = 2T(n/2) + P_{\mathcal{F}}(n)f(\sqrt{n}).$$

□

3. Data Structures

We have shown how to use sparsification to speed up structurally kinetic minimum spanning tree data structures. We now describe some techniques for constructing these data structures.

As our kinetic or parametric algorithm progresses, the minimum spanning tree it maintains will change by *swaps*, in which one edge is removed and another inserted. We begin by giving a geometric interpretation for these swaps. We then partition the vertices into clusters, and classify the swaps according to the inserted location of the endpoints of the edge in the clusters, and show how to find the next-occurring swap within each class.

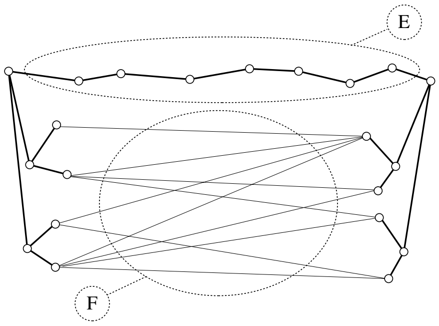

Figure 1. Two sets of edges such that every pair of one edge from each set forms a swap.

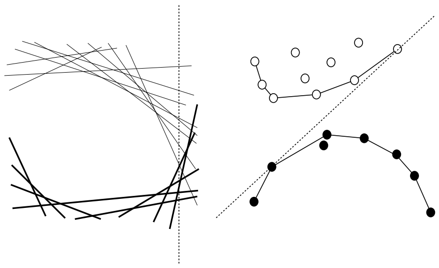

Figure 2. First non-positive swap: in line arrangement (left), rightmost point above lines from tree edges and below lines from non-tree edges; in dual point arrangement (right), line with highest slope above points from tree edges and below points from non-tree edges.

3.1. Swaps, Duality, and Bitangents

As our algorithm progresses, topology changes arising from insertions and deletions of edges will be relatively easy to maintain. However, it will require work to locate topology changes arising from changes in relative ordering of edge weights. For a given spanning tree T of $\mathcal{G}$ and two edges $e, f \in \mathcal{E}$, we say that e and f form a *swap* if $e \in T$, $f \notin T$, and the cycle induced by f in T contains the edge e. For any fixed value of λ, define the *weight* of a swap (e, f), denoted $\Delta_{e,f}(\lambda)$, to be $w_f(\lambda) - w_e(\lambda)$; if e and f do not form a swap, we set $\Delta_{e,f}(\lambda) = +\infty$. That is, $\Delta_{e,f}(\lambda)$ is the amount by which the tree weight would increase if the swap were performed. Given a value λ_0 and $MST(\lambda_0)$, our algorithm will need to find the first value of $\lambda \geq \lambda_0$ for which $\Delta_{e,f}(\lambda) \leq 0$, for some pair $e, f \in \mathcal{E}$. For a pair $E, F \subseteq \mathcal{E}$, where $E \subset MST(\lambda_0)$ and $F \cap MST(\lambda_0) = \emptyset$, we define the *next swap* (or *first non-positive swap*) between E and F to be the pair $e \in E$ and $f \in F$ so that $\Delta_{e,f}(\lambda^*) \leq 0$ for some $\lambda^* \geq \lambda_0$ and $\Delta_{e',f'}(\lambda) > 0$ for all pairs $e' \in E$, $f \in F$ and for all $\lambda_0 \leq \lambda < \lambda^*$. The pair of edges $g \in E, h \in F$ for which $\Delta_{g,h}(\lambda_0)$ has the minimum value is called the *best swap* at λ_0.

To help understand the problem of computing the next swap, we interpret swaps geometrically. Suppose we have a subset E of edges of a spanning tree of $\mathcal{G}$ and a subset F of edges not in the spanning tree, where every pair (e, f) of one edge from each set forms a swap (Figure 1). We can form a line arrangement in the (λ, w) plane, where each line is the graph of the weight function of a single edge. The first non-positive swap can be found as the first (leftmost) point in the arrangement where a line from F crosses below a line from E; or, equivalently it is the last (rightmost) point that lies on or below all lines from F and on or above all lines from E (Figure 2).

We apply a projective duality to this configuration, in which each line $w = a\lambda + b$ in the primal (λ, w) plane is transformed into a point $(-a, b)$ in the dual (x, y) plane, and each point (λ, w) in the primal plane is transformed into a line $y = \lambda x + w$ in the dual plane. This transformation preserves point-line incidences and above-below relationships. The dual transform maps the graph of the weight of each edge e to a point. For a subset $X \subseteq \mathcal{E}$, let S_E denote the set of such points corresponding to the edges in X. In the dual plane, the first non-positive swap corresponds to the maximum-slope line that lies on or below all points in S_F and on or above all points in S_E. Such a line is a *bitangent* to the lower convex hull of S_F and the upper convex hull of S_E (Figure 2(b)).

Because of this connection between non-positive swaps and bitangents, we can apply computational geometry techniques in our solution of the parametric and kinetic minimum spanning tree problems. If we are given a representation of the two hulls above that supports binary searches, their bitangent can be found in $O(\log n)$ time. In the special case in which E or F consists of a single edge, we are simply seeking a tangent through the corresponding dual point to the convex hull of the points corresponding to the other set, and again this can be done in logarithmic time.

3.2. Dynamic Convex Hull Data Structure

Because of the connection between swaps and hulls outlined above, our algorithm will need to use some data structures for maintaining convex hulls of point sets. The specific operations we need are point insertion and *undo* operations: an undo deletes the most recently inserted point remaining in the data structure.

Theorem 1. *We can maintain the convex hull of a planar point set, subject to insertions and undo operations, in time $O(\log n)$ per update or query.*

Proof: We maintain a sorted list of the vertices on the hull using any of various balanced binary search tree data structures. The time bound for queries then becomes immediate. To insert a point, we do two binary search queries to locate its left and right tangents, split the sorted list of hull vertices at those two points, and rejoin the left side of the left cut, the new vertex, and the right side of the right cut. We retain pointers to the discarded subtrees so that undo operations can similarly be performed by $O(1)$ split and join operations. □

3.3. Parametric Search

In several cases of our algorithm it will prove easier to find the best swap for a fixed value of λ than to find the value λ^* leading to the first non-positive swap. Megiddo's parametric search [26] provides a general mechanism for turning an algorithm for the former problem into an algorithm for the latter.

The parametric search method starts from two given algorithms: a *decision oracle* that determines if a given λ is less or greater than λ^*, and a *simulated algorithm* that computes a function $f(\lambda)$ discontinuous at λ^*. The conditional branches of the simulated algorithm must depend only on low-degree polynomials in λ. Since the decision oracle is discontinuous at λ^*, it is common to use the same algorithm in both roles. Parametric search then produces the sequences of steps the simulated algorithm would perform if it were given λ^* as its argument; each conditional branch is simulated by using the decision algorithm to compare λ^* with the roots of the polynomial tested at that branch. Because of the simulated algorithm's discontinuity, we must eventually find a root equal to λ^*. If the decision oracle takes time T_D, and the simulated algorithm is a parallel algorithm taking time T_S with P_S processors, we can test many roots at once using binary search, giving an overall time of $O(T_D T_S \log P_S + T_S P_S)$. Standard techniques for speeding this up further include moving as much as possible out of the simulated algorithm, using partial results to speed up the decision oracle, and Cole's technique for avoiding the $\log P_S$ factor by allowing a constant fraction of the simulated processors to fail to make progress at each step [6].

3.4. Restricted Partitions

Our algorithms use a technique of partitioning trees and forests into smaller subtrees, or clusters of vertices, that was introduced by Frederickson [15, 16] and used by him and others as part of various dynamic graph algorithms. We will combine this clustering technique with some geometric data structures (primarily, planar convex hulls) in a manner similar to techniques used in our previous paper on speedups in the network simplex method [10].

We first transform our input graph $\mathcal{G}$ into a new graph $\mathcal{G}'$ with degree at most three, so any tree in $\mathcal{G}'$ will be binary. Let v be any edge of degree $\Delta > 3$; replace v by $\Delta - 2$ vertices connected by a path. Path endpoints receive two of the original edges of v, and each interior vertex receives one. Path edges are given a cost function that is a sufficiently small constant so that all path edges are always part of the current minimum spanning tree. This transformed graph is not hard to maintain as $\mathcal{G}$ undergoes edge insertions or deletions: each update in $\mathcal{G}$ causes a constant number of updates to $\mathcal{G}'$. Thus, for the remainder of the description of our kinetic algorithm, we assume our input graph has all vertex degrees at most three.

The following definition is due to Frederickson [16].

Definition 1. A restricted partition of order z with respect to a tree T in which all vertex degrees are at most three is a partition of the vertices of V such that:

1. Each set in the partition contains at most z vertices.
2. Each set in the partition induces a connected subtree of T.
3. For each set S in the partition, if S contains more than one vertex, then there are at most two tree edges having one endpoint in S.
4. No two sets can be combined and still satisfy the other conditions.

We call each set in the partition a *cluster*. The endpoints of an edge of T connecting two different clusters are called *terminal vertices*. Each cluster has at most two terminal vertices. A cluster with k terminal vertices will be referred to as a *k-terminal cluster*.

Frederickson also showed that such a partition can easily be found in linear time. There are at most $O(n/z)$ clusters in a restricted partition of an n-vertex tree. If we change the tree by performing a swap, we can update the restricted partition in time $O(z)$ by splitting and re-merging $O(1)$ clusters [16].

Given a restricted partition of the current minimum spanning tree in a parametric or kinetic MST problem, we can classify the potential swaps into three types according to how many clusters are involved in the endpoints of the swapped edges:

Definition 2. *Let edges e and f form a swap in a tree for which we have a restricted partition, so that e is a tree edge on the tree path between the endpoints of f. Then if both endpoints of f are in a single cluster, e must be in the same cluster as f; we call this an* intra-cluster swap. *If f has endpoints in different clusters, and e belongs to one of these two clusters, we call this a* dual-cluster swap. *Finally, if the endpoints of f do not lie in the same cluster and e does not belong to one of these two clusters, we call this an* inter-cluster swap.

We next show how to maintain the next swap that can occur, for each of these three types of swap.

3.5. Intra-Cluster Swaps

To find the first non-positive intra-cluster swap in a given cluster, we apply parametric search. Recall that this requires two subroutines: a *decision oracle* for comparing a given parameter λ to the optimal value λ^* we are seeking, and a *simulated algorithm* discontinuous at λ^*.

Lemma 9. *A decision oracle for the first non-positive intra-cluster swap in a cluster of $O(z)$ vertices can be implemented in time $O(z)$.*

Proof: We need to detect whether the given value of λ is before or after the first non-positive swap; equivalently, whether there exists a non-positive swap at λ itself. Since a given spanning tree is the minimum only within a single interval of values of λ, we perform this test by computing the values of all intra-cluster edges at the parameter value λ, and testing whether the given tree is still the minimum spanning tree at that parameter using a minimum spanning tree verification algorithm [8, 25]. □

Lemma 10. *We can find the first non-positive intra-cluster swap in a cluster of $O(z)$ vertices in $O(z \log z)$ time.*

Proof: We apply parametric search with the decision oracle described above. For a simulated algorithm, we use sorting, since the sorted order is discontinuous at all swaps. Cole [6] shows how to apply parametric search to sorting with $O(\log n)$ calls to the decision oracle and $O(n \log n)$ additive overhead. □

3.6. Dual-Cluster Swaps

To find dual-cluster swaps, we combine the dynamic convex hull data structure described earlier with Frederickson's idea of *ambivalent data structures*. Let C be a cluster in a restricted partition. A non-tree edge whose one endpoint lies in C and the other does not lie in C is called an *external* edge of C. For each cluster C, we want to maintain the next dual swap that involves an external edge of C. Recall that C has at most two terminal vertices. An external edge of C incident upon a vertex u of C can swap only with an edge of C that lies on the path from u to a terminal vertex of C. For each such external edge and each terminal vertex v of C, we will therefore store the edge e on the path from u to v for which $w_f(\lambda) - w_e(\lambda)$ becomes zero first. Let us denote this edge by $\mu_v(f)$. Note that if v lies on the path in the minimum spanning tree between the endpoints of f, then $(\mu_v(f), f)$ forms a swap.

Lemma 11. *Let Π be a restricted partition of order z, and let C be a newly formed cluster of Π. For all non-tree* *edges f having exactly one endpoint in C and for each terminal vertex v of C, we can compute $\mu_v(f)$ in total time $O(z \log z)$.*

Proof: Let v be a terminal vertex of C. We traverse the subtree contained in C, starting from v, and keep track of the edges on the path from the current vertex u to v. As outlined in Section 3.1 the weights of the edges in this path correspond to points in a plane, and we use the dynamic convex hull data structure described in Theorem 1 to maintain the convex hull of these points. When our traversal first visits an edge of the tree we insert the corresponding point into the hull, and when we return from traversing an edge we perform an undo operation to remove the point from the hull. Then, as described in Section 3.1, for an external non-tree edge f incident upon u, $\mu_v(f)$ can be found by computing a tangent from the point corresponding to f's weight function to the current hull, in time $O(\log z)$. □

Lemma 12. *As we dynamically maintain a restricted partition of order z on a spanning tree of a dynamic m-vertex degree-three graph, we can maintain a data structure in time $O(z \log z + m/z)$ per update to the graph which will let us query the first non-positive dual-cluster swap in time $O(1)$ per pair of clusters.*

Proof: We store for each endpoint of each non-tree edge the information described in Lemma 11. Since each update modifies only $O(1)$ clusters, we can recompute this information in $O(z \log z)$ per update, as described in that lemma. For each cluster C, we partition the external edges in C into $O(m/z)$ groups, so that all edges whose other endpoints lie in the same cluster $C_i \neq C$ belong to the same group. For each such group F_i and for each terminal vertex v of C, we store the pair $\mu_v(C_i) = (\mu_v(f), f)$ for which $w_f - w_{\mu_v(f)}$ becomes zero first among all edges $f \in F_i$. This information can be updated in time $O(z)$ whenever a cluster is modified. We also store a lowest-common-ancestor data structure for the tree formed by contracting each cluster of the partition; this takes time $O(m/z)$ per update to maintain.

Suppose we want to find the next dual swap involving an external edge whose one endpoint lies in C_1 and the other in C_2. We first find the terminal vertices v_1, v_2 of C_1, C_2, respectively, that connect the path from C_1 to C_2 in the minimum spanning tree. This can be done in $O(1)$ time using the lowest-common-ancestor data structure. It is easily seen that $\mu_{v_1}(C_2)$ and $\mu_{v_2}(C_1)$ form swaps. Of these two, we return the one whose weight becomes zero first. □

3.7. Inter-Cluster Swaps

We now need to show how to find the first non-positive inter-cluster swap. We first describe a deterministic technique based on the intra-cluster swap technique described in Section 3.5.

Recall that we will be maintaining a restricted partition of order z of the minimum spanning tree of the graph. For any value of λ, consider forming the following contracted graph $\mathcal{G}'(\lambda)$ from $\mathcal{G}$, with only $O(m/z)$ vertices: Contract each 1-terminal cluster, with the terminal vertex v, to a single node v. Contract each 2-terminal cluster, with terminal vertices u, v, to an edge (u, v) whose weight is equal the weight of the heaviest edge on the path connecting u and v. Let C_1, C_2 be two clusters, and let v_1 and v_2 be the terminal vertices of C_1, C_2, respectively, that lie on the path connecting C_1 to C_2. We contract all non-tree edges between C_1 and C_2 to a single edge (v_1, v_2) whose weight is equal to the lightest weight among all the non-tree edges between C_1 and C_2.

Lemma 13. *We can maintain a data structure in time $O(z \log z)$ per change to the restricted partition, so that for any λ, the graph $\mathcal{G}'(\lambda)$ described above can be found in time $O(\log n)$ per edge.*

Proof: Suppose we maintain a restricted partition of order z of the minimum spanning tree of $\mathcal{G}$. We can maintain the convex hull of the points corresponding to the path connecting the two terminals of each two-terminal cluster, and of the points corresponding to the edges connecting each pair of clusters. The weight of each edge in $\mathcal{G}'$ can be found in time $O(\log n)$ by binary search in the appropriate hull. □

Lemma 14. *The first non-positive inter-cluster swap in $\mathcal{G}$ is the first non-positive swap in $\mathcal{G}'(\lambda^*)$.*

Lemma 15. *We can maintain a data structure in time $O(z \log z)$ per change to the restricted partition, such that if the graph $\mathcal{G}'(\lambda)$ described above has m' edges, the best inter-cluster swap can be found in time $O(m' \log^2 n)$.*

Proof: The structure we maintain is simply the set of hulls described in Lemma 13. To find the best swap, we apply a parametric search routine similar to the one described in Section 3.5. We modify the minimum spanning tree verification used as the decision oracle, to compute $\mathcal{G}'$ and then verify that the contraction of the current spanning tree is the true minimum spanning tree of $\mathcal{G}'$; this takes time $O(m' \log n)$ per oracle call. We also modify the simulated algorithm, to compute G' before sorting its edge weights; the computation of $\mathcal{G}'$ is just a collection of parallel binary searches and does not increase the overall complexity beyond its previous bound of $O(\log m')$ oracle calls and $O(m' \log m')$ additive overhead. □

With the use of randomization, we can reduce this bound slightly:

Lemma 16. *We can maintain a data structure in time $O(z \log z)$ per change to the restricted partition, such that if the graph $\mathcal{G}'(\lambda)$ described above has m' edges, the best inter-cluster swap can be found in expected time $O(m' \log n)$.*

Proof: We perform two different cases, depending on whether the contracted graph $\mathcal{G}'$ is sparse or not. If it has at least as many non-tree edges as tree edges, we choose randomly a non-tree edge e of $\mathcal{G}'$, find the best swap involving that edge and one of the tree edges, and use this swap to eliminate (in expectation) half the non-tree edges. Alternately, if $\mathcal{G}'$ has few non-tree edges, we choose a tree edge e randomly, and use the best swap involving that edge to eliminate in expectation half the tree edges. Repeating this process eventually leads to an empty graph, at which point we return the best swap found in the process. We omit the details in this extended abstract. □

4. Parametric and Kinetic MST Algorithms

The data structures described in the preceeding sections let us find the next non-positive swap of each type. We are ready to put them together into our overall kinetic minimum spanning tree algorithm.

4.1. General Graphs

Theorem 2. *We can maintain a graph, having at most m linearly weighted edges at any one time, and keep track of the minimum spanning tree kinetically, in time $O(pm^{2/3} \log^{4/3} m)$, where p denotes the number of edge insertions, edge deletions, or minimum spanning tree topology changes. If we allow randomization the expected time is $O(pm^{2/3} \log m)$.*

Proof: We apply the transformation described above to make $\mathcal{G}$ have degree at most three, which increases the number of nodes to $O(m)$. Then we use the data structures described above to keep track of the next non-positive swap, in time $O(z \log z + m/z + (m/z)^2 \log^2 z)$ per update. When we encounter an edge insertion, we update these structures, and use them to test whether a non-positive swap exists at the time of insertion; if so we perform the swap. When we encounter a deletion of a minimum spanning tree edge, we use our convex hull data structures to find the best replacement edge in each group of edges connecting a pair of clusters; there are $O((m/z)^2)$ such groups, so this step takes time $O((m/z)^2 \log n)$. When we encounter a deletion of a non-tree edge we update our structures and continue. And, when the next non-positive swap found after one update occurs before the next insertion or deletion operation, we again update our structures and continue. Setting $z = m^{2/3} \log^{1/3} n$ or $m^{2/3}$ produces the bound above. □

We now apply sparsification to further improve these bounds.

Theorem 3. *We can solve the kinetic minimum spanning tree problem in time $O(K(n, m)n^{2/3} \log^{4/3} n)$ or in randomized expected time $O(K(n, m)n^{2/3} \log n)$. We can solve the parametric minimum spanning tree problem in time $O(P(n, m)n^{2/3} \log^{4/3} n)$ or in randomized expected time $O(P(n, m)n^{2/3} \log n)$.*

4.2. Planar and Minor-Closed Graph Families

Our time bound becomes better for planar graphs or other minor-closed families of graphs, because the contracted graph $\mathcal{G}'$ is sparse.

Theorem 4. *We can maintain a graph, having at most n vertices at any one time, belonging to some minor-closed family $\mathcal{F}$, and keep track of the structurally kinetic minimum spanning tree, in total time $O(pn^{1/2} \log^{3/2} n)$. If we allow randomization the expected total time is $O(pn^{1/2} \log n)$.*

Proof: Because the number of non-tree edges in $\mathcal{G}'$ is $O(n/z)$, the tradeoff above reduces to $O(z \log z + (n/z) \log^2 z)$, or randomized $O(z \log z + (n/z) \log z)$. Setting z to $n^{1/2} \log^{1/2} n$ or $n^{1/2}$ produces the stated bounds. □

Again, applying sparsification leads to further improvements.

Theorem 5. *We can maintain a graph, having at most n vertices at any one time, belonging to some minor-closed family F, and keep track of the parametric or functionally kinetic minimum spanning tree, in total time $O(n^{3/2} + P(n)n^{1/4} \log^{3/2} n)$. If we allow randomization the expected total time is $O(n^{3/2} + P(n)n^{1/4} \log n)$. For planar graphs the $O(n^{3/2})$ term can be removed from these bounds.*

With Dey's bound $P(n) = O(n^{4/3})$, our total time is worst-case bounded by $O(n^{19/12} \log^{3/2} n)$, or randomized expected time $O(n^{19/12} \log n)$.

5. Conclusions

We have given deterministic and randomized algorithms for solving the parametric and kinetic minimum spanning tree problems for general graphs, and improvements for special families, such as minor-closed and planar graphs. The mixture of graph-theoretic and geometric attributes is an especially appealing aspect of this problem.

It would be desirable to find kinetic data structures for maintaining the MST of a graph that do not require the heavy arsenal of tools we have used: sparsification, both general and separator-based, dynamic convex hulls, restricted partitions, ambivalent data structures, and parametric search. We plan to work both on simplifying our methods and on improving our bounds.

References

[1] P. K. Agarwal and M. Sharir. Algorithmic techniques for geometric optimization. In *Computer Science Today: Recent Trends and Developments*, Lecture Notes in Computer Science, vol. 100 (J. van Leeuwen, ed.), Springer-Verlag, 1995.

[2] N. Alon, P. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proc. 22nd ACM Symp. Theory of Computing*, 1990, 293–299.

[3] J. Basch, L. J. Guibas, and J. Hershberger., pp. 293–299 Data structures for mobile data. In *Proc. 8th ACM-SIAM Symp. Discrete Algorithms*, 1997, 747–756.

[4] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proc. 13th ACM Symp. Computational Geometry*, 1997, 344–351.

[5] B. Chazelle. A faster deterministic algorithm for minimum spanning trees. In *Proc. 38th Symp. Foundations of Computer Science*, 1997, 22–31.

[6] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34 (1987), 200–208.

[7] T. K. Dey. Improved bounds on planar k-sets and k-levels. *Discrete & Computational Geometry*. 19 (1998), 373–382.

[8] B. Dixon, M. Rauch, and R. E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. Computing*, 21 (1992), 1184–1192.

[9] D. Eppstein. Geometric lower bounds for parametric matroid optimization. To appear in *Discrete & Computational Geometry*.

[10] D. Eppstein. Clustering for faster network simplex pivots. In *Proc. 5th ACM-SIAM Symp. Discrete Algorithms*, 1994, 160–166.

[11] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification — A technique for speeding up dynamic graph algorithms. *J. ACM*, 44 (1997), 669–696.

[12] D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer. Separator based sparsification I: planarity testing and minimum spanning trees. *J. Computing & Systems Sciences*, 52 (1996), 3–27.

[13] D. Fernández-Baca and G. Slutzki. Parametric problems on graphs of bounded tree-width. *J. Algorithms*, 16 (1994), 408–430.

[14] D. Fernández-Baca, G. Slutzki, and D. Eppstein. Using sparsification for parametric minimum spanning tree problems. *Nordic J. Computing*, 3 (1996), 352–366.

[15] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees. *SIAM J. Computing*, 14 (1985), 781–798.

[16] G. N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *SIAM J. Computing*, 26 (1997), 484–538.

[17] M. T. Goodrich. Planar separators and parallel polygon triangulation. *J. Computing & Systems Sciences*, 51 (1995), 374–389.

[18] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Ann. Hist. Comput.*, 7 (1985), 43–57.

[19] L. J. Guibas. Kinetic data structures — a state of the art report. To appear in *Proc. 3rd Worksh. Algorithmic Foundations of Robotics*, 1998.

[20] M. R. Henzinger and V. King. Maintaining minimum spanning trees in dynamic graphs. In *Proc. 24th Int. Coll. Automata, Languages and Programming*, Lecture Notes in Computer Science, vol. 1256, Springer-Verlag, 1997, 594–604.

[21] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity and minimum spanning tree. In *Proc. 30th ACM Symp. Theory of Computing*, 1998, 79–89.

[22] D. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. *J. ACM*, 42 (1995), 321–329.

[23] N. Katoh. Parametric combinatorial optimization problems and applications. *J. Inst. Electronics, Information and Communication Engineers*, 74 (1991), 949–956.

[24] N. Katoh. Bicriteria network optimization problems. *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, E75-A (1992), 321–329.

[25] V. King. A simpler minimum spanning tree verification algorithm. In *Proc. 4th Worksh. Algorithms and Data Structures*, Lecture Notes in Computer Science, vol. 955, Springer-Verlag, 1995, 440–448.

[26] N. Megiddo. Applying parallel computation algorithms in the design of sequential algorithms. *J. ACM*, 30 (1983), 852–865.

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

[28] N. E. Young, R. E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum-balance algorithms. *Networks*, 21 (1991), 205–221.