Finding the Shortest Bottleneck Edge in a Parametric Minimum Spanning Tree

Timothy M. Chan*

The result. *Parametric optimization problems* that concern graphs with continuously changing edge weights have been explored by numerous researchers, with motivation ranging from sensitivity analysis to mobile-data applications. For instance, Dey [5] has shown that for an undirected graph with n vertices and m edges where the edge weights are linear functions in one parameter ("time"), the minimum spanning tree (MST) can undergo at most $O(mn^{1/3})$ changes (edge swaps). Agarwal *et al.* [1] have given data structures to maintain the MST over time, with a cost of $O(n^{2/3} \operatorname{polylog} n)$ per change. Fernandez-Baca *et al.* [7] have given an algorithm to compute all changes to the MST in $O(mn \log n)$ total time.

In this note, we focus on a problem studied by Katoh and Tokuyama [8]:

> Given a parametric graph with edge weights changing linearly in time, find the time value when the weight of the largest MST edge (the so-called *bottleneck edge*) is minimized.

The bottleneck edge weight is of particular interest, because it represents a threshold for connectivity: it is equal to the smallest value r such that the subgraph of edges with weight $\leq r$ stays connected.

For this problem, Katoh and Tokuyama [8] have given an $O((m^{8/7}n^{1/7} + mn^{1/3}) \operatorname{polylog} n)$ algorithm, which is faster than the current methods for computing all MSTs over time. Katoh and Tokuyama's method uses advanced data structures for range searching and is therefore difficult to implement. Here, we give a much faster and simpler randomized algorithm that runs in $O(n(m/n)^{\varepsilon} \log n + m)$ expected time for any fixed $\varepsilon > 0$. This time bound is at least as good as $O(m \log n)$ and $O(n \log^{1+\varepsilon'} n + m)$ for any fixed $\varepsilon' > 0$, and almost matches an $\Omega(n \log n + m)$ lower bound. The new result is obtained by an interesting combination of techniques from computational geometry and graph data structures. The geometric aspects are similar to those used for the problem of 2-d feasible linear programming with violations from a previous paper [4].

*School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, `tmchan@uwaterloo.ca`. This work was supported in part by NSERC.

Randomized reduction. Let $G = (V, E)$ be the given parametric graph. Let $G(t)$ denote the graph at time t. Let $G(t, r)$ denote the (unweighted) subgraph consisting of all edges of $G(t)$ with weight $\leq r$. Our problem is to find the smallest r such that $G(t, r)$ is connected for some t.

We adopt a geometric approach: consider the region in the plane $\Gamma_G = \{(t, r) \in \mathbb{R}^2 \mid G(t, r) \text{ is connected}\}$. The boundary of this region is an x-monotone chain inside an arrangement of lines $\{\ell_e\}_{e \in E}$, where ℓ_e denotes the graph of the weight function of edge e. We slightly extend the problem: given a parametric (multi)graph G and a triangle Δ in the plane, compute $w(G, \Delta)$, the lowest y-coordinate in the region $\Gamma_G \cap \Delta$.

Our first step is to reduce this optimization problem to a decision problem using the author's randomized technique [4]. In order to apply the technique, we need to show how to form a constant number of subproblems, each of a fraction of the original size, such that the answer to the original problem can be expressed as the minimum of the answers to the subproblems.

This can be done as follows: first compute a *cutting* [3] of Δ into a constant number of triangles Δ_i, such that each Δ_i intersects at most a fraction of the lines in $\{\ell_e\}_{e \in E}$. Consider each triangle Δ_i. Classify edges into three types: let $E_i = \{e \in E \mid \ell_e \text{ intersects } \Delta_i\}$, $E_i^+ = \{e \in E \mid \ell_e \text{ is completely above } \Delta_i\}$, and $E_i^- = \{e \in E \mid \ell_e \text{ is completely below } \Delta_i\}$. As long as $(t, r) \in \Delta_i$, edges in E_i^+ can never be present in $G(t, r)$, while edges in E_i^- are always present in $G(t, r)$. In deciding whether $G(t, r)$ is connected for $(t, r) \in \Delta_i$, we can thus consider a smaller subgraph G_i of G, formed by removing all edges in E_i^+ and contracting all edges in E_i^- (i.e., condensing each connected component of (V, E_i^-) into a single vertex). The number of edges in G_i is $|E_i|$, which is at most a fraction of m, and the construction takes $O(m)$ time. Since $w(G, \Delta) = \min_i w(G, \Delta_i) = \min_i w(G_i, \Delta_i)$, the requirement of the randomized reduction technique is established.

Off-line dynamic connectivity. We now solve the decision problem: given a parametric (multi)graph G, a value r, and a triangle Δ, determine whether $w(G, \Delta) \leq r$. Let $\Delta' = \Delta \cap \{(x, y) \mid y \leq r\}$. Note that the answer is yes iff Γ_G intersects Δ', or equivalently, Γ_G intersects the boundary of Δ' (which consists of at most four line segments).

We can thus reduce the decision problem to the problem of intersecting Γ_G with a given line ℓ, i.e., determining when $G(t,r)$ is connected over $(t,r) \in \ell$. This is a *dynamic* graph connectivity problem: as we travel from left to right on the line ℓ, an edge e is inserted to or deleted from $G(t,r)$ whenever we pass a line ℓ_e from above or below. Immediately, known polylogarithmic connectivity data structures give us an $O(m \operatorname{polylog} n)$ time bound [9]. However, because the insertion/deletion time value of each edge e (i.e., the x-coordinate of the intersection of ℓ_e with ℓ) is known in advance, less powerful *off-line* data structures would suffice in our case.

For instance, Eppstein [6] has solved the off-line version of the (harder) dynamic minimum spanning tree problem in $O(\log n)$ time per update. A simpler $O(\log n)$ method for off-line dynamic connectivity can also be obtained directly by divide-and-conquer over the update sequence (we omit the details here). We conclude that the decision problem can be solved in $O(m \log n)$ time, and by the randomized reduction technique [4], the optimization problem can be solved in $O(m \log n)$ expected time.

For dense graphs, we can speed up the decision algorithm by exploiting the following observation: in our application of dynamic connectivity, an edge can either be inserted or deleted, but not both. Our idea is to "sparsify" the graph first. Interestingly, this sparsification is accomplished by computing MSTs under a different set of weights:

For the subgraph G^+ of edges with insertion time values, compute the minimum spanning forest F^+ with the insertion times as weights. For the subgraph G^- of edges with deletion time values, compute the maximum spanning forest F^- with the deletion times as weights. (Prim's algorithm with Fibonacci heaps is good enough here.) We solve the off-line connectivity problem on the subgraph $F = F^+ \cup F^-$, which has only $\leq 2(n-1)$ edges, in $O(n \log n)$ time. To see that the solution remains correct, consider an edge $uv \notin F$; w.l.o.g., say $uv \in G^+$. By a well-known property of minimum spanning forests, uv has a larger weight than all edges along the path π from u to v in F^+. So, at the insertion time of uv, the path π is already present in F, making the edge uv unnecessary. We conclude that the decision problem can be solved in $O(n \log n + m)$ time.

In applying the randomized reduction technique, we need to weaken the decision time bound to $T(n,m) = O(n^{1-\varepsilon} m^{\varepsilon} \log n + m)$, so that $T(n,m)/m^{\varepsilon}$ is increasing [4]. We conclude that the optimization problem can be solved in $O(n(m/n)^{\varepsilon} \log n + m)$ expected time.

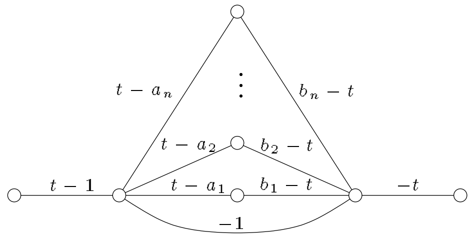

Figure 1: The bottleneck edge weight is ≤ 0 iff $t \in [0,1] \setminus ((a_1,b_1) \cup \cdots \cup (a_n,b_n))$.

Remarks. An $\Omega(n \log n + m)$ lower bound follows by a reduction, illustrated in Figure 1, from the following problem: given n intervals in the real line, decide whether their union covers $[0,1]$. This problem requires $\Omega(n \log n)$ time in the algebraic decision tree model [2].

Our algorithm can be easily modified for a related parametric graph problem: minimize the cost of the *bottleneck shortest path* between two given vertices, where the cost of a path here is its largest edge weight. The $O(m \log n)$ version of our algorithm works under more general, nonlinear weight functions. Other objective functions can also be handled.

References

[1] P. K. Agarwal, D. Eppstein, L. J. Guibas, and M. R. Henzinger. Parametric and kinetic minimum spanning trees. In *Proc. 39th IEEE Sympos. Found. Comput. Sci.*, pages 596–605, 1998.

[2] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th ACM Sympos. Theory Comput.*, pages 80–86, 1983.

[3] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, Germany, 2000.

[4] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22:547–567, 1999.

[5] T. K. Dey. Improved bounds on planar k-sets and related problems. *Discrete Comput. Geom.*, 19:373–382, 1998.

[6] D. Eppstein. Offline algorithms for dynamic minimum spanning tree problems. *J. Algorithms*, 17:237–250, 1994.

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

[8] N. Katoh and T. Tokuyama. Notes on computing peaks in k-levels and parametric spanning trees. In *Proc. 17th ACM Sympos. Comput. Geom.*, pages 241–251, 2001.

[9] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd ACM Sympos. Theory Comput.*, pages 343–350, 2000.