Centrality problems on dynamic trees

Stephen Alstrup* Jacob Holm* Kristian de Lichtenberg*

December 9, 1997

Abstract

Sleator and Tarjan's dynamic trees have proved an excellent tool. Specifically, in cases where one wish to maintain local properties, such as "the minimum edge weight on a path". The dynamic trees have also been used to maintain nodes with global properties such as the 1-median (V. Auletta, D. Parente and G. Persiano, TCS'96) and 1-center (S. Cheng and M. Ng, SODA'96), but in these cases the time complexity per operation becomes $O(\log^2 n)$ compared to the usual $O(\log n)$ for maintaining local properties. Furthermore the algorithm becomes rather complicated. In this paper we show how topology trees can provide a simple tool for maintaining global and local properties with complexity $O(\log n)$ per operation. As examples, we show how to maintain 1-center and 1-median in fully dynamic trees with complexity $O(\log n)$ per operation.

1 Introduction

In this paper we investigate how to maintain global properties of dynamic trees. Specifically, we consider the problem of finding a node/edge which is "most central" with respect to a given cost criterion. We present a powerful black box which can be used on this kind of optimization problems.

In order to demonstrate the black box we consider the dynamic 1-median and 1-center problems. In 1971 Goldman [9] gave a linear time algorithm for determining a node in a tree, called a *1-median*, minimizing the sum of the weighted distances to all other nodes. In 1973 Handler [11] showed how one in linear time can compute a *1-center* of a tree, minimizing the maximal distance to any other node. The static median and center problems have been investigated and generalized in many papers, see e.g. [10, 2, 8, 5]. A long list of references to the 1-median and 1-center problem and similar problems can be found in [12].

*E-mail: (stephen,samson,morat)@diku.dk. Department of Computer Science, University of Copenhagen.

More specific, the 1-median problem is defined as follows. Let V be the weighted nodes in the tree. The task is to choose a node $\mathcal{M}$, such that $\sum_{v \in V} weight(v) * dist(v, \mathcal{M})$ is minimal, where $dist(v, \mathcal{M})$ is the sum of *cost* of edges on the unique path from v to $\mathcal{M}$ in the tree. Both *cost* and *weight* are assumed to be nonnegative. In [2] Auletta, Parente and Persiano showed how to find a 1-median in time $O(\log^2 n)$ in a tree after a change of a node weight.

The 1-center problem is defined as follows. Let V be the nodes in the tree. The task is to choose a node $\mathcal{C}$, such that $\max_{v \in V} dist(v, \mathcal{C})$ is minimal, where *cost* again is assumed to be nonnegative. In [3] Cheng and Ng showed how to maintain the 1-center for a dynamic forest under *link* and *cut* in $O(\log^2 n)$ time.

Both of these algorithms use the dynamic trees by Sleator and Tarjan [13]. Sleator and Tarjan's dynamic trees have proved an excellent tool for finding nodes or edges such as "the minimum edge weight on a path". For this kind of problems we have the following nice local property: *Let T be a tree and let S be a subtree of T. For any $x \in S$ we have that x is a solution to T implies that x is a solution to S.* The difficulty in solving the 1-center and 1-median problem comes from the fact that we do not have this property for those problems.

Here we show how to maintain the 1-center and 1-median in $O(\log n)$ worst case per operation including *link*, *cut*, and *change* of edge/node weight. Both results follows as simple applications of the black box. The black box is presented in section 3 and builds on the topology trees which are shortly presented in section 2. In section 4 the applications are given.

Combining the results from [1, 7] with the results in this paper shows how topology trees can be used as a black box to maintain global and local properties and for searching in dynamic trees.

2 Topology Trees

In this section we give a short presentation of the topology trees by Frederickson [6, 4]. Our presentation differ slightly from the original topology trees, since we use the simpler and less restrictive version as defined in [1]

Let T be an arbitrary tree with n nodes. For a connected subtree of T, we call a node which has edges out of the subtree a *boundary node*. A *cluster* is a connected subtree of T with at most two boundary nodes. The set of boundary nodes of a cluster C is denoted ∂C. We say that $\partial C = \{a, b\}$ if C has boundary nodes a and b even if a and b are identical. Two clusters are said to be *neighbors* if they intersect in exactly one node. A *topology tree* $\mathcal{T}$ of T is a binary tree such that:

1. The nodes of $\mathcal{T}$ represents clusters of T.
2. The leaves of $\mathcal{T}$ represents the edges of T.

3. If C is represented by an internal node of $\mathcal{T}$ with children representing A and B, then $C = A \cup B$ and A and B are neighbors.
4. The root of $\mathcal{T}$ represents T.
5. The height of $\mathcal{T}$ is $O(\log n)$.

A tree with a single node has an empty topology tree. From [1] we have the following theorem.

Theorem 1 *Let info be some information of clusters in a dynamic forest with n nodes so that*

1. *For any edge e, info($\{e\}$) can be computed in time t_1.*
2. *For any neighboring clusters C_1 and C_2, info($C_1 \cup C_2$) can be computed in time t_2, given info(C_1) and info(C_2).*

Then we can maintain info for all trees in a dynamic forest in $O(t_1 + t_2 \log n)$ worst case time per link and cut, given the ability to use $O(n * (t_1 + t_2))$ time and $O(n)$ space for preprocessing. □

3 Black box for finding edges and nodes with global properties

In this section we provide a general tool for maintaining nodes and edges with global properties in dynamic trees. We give a general search algorithm which require:

1. Given two neighboring clusters which together represents the *whole* tree and some related information we can decide to which cluster the node or edge requested belongs.
2. The information for an *arbitrary* cluster should be efficiently computable by merging information of sub-clusters.

In the following section we will see examples on how to use the tool.

Theorem 2 *Let info be some information of clusters in a dynamic forest with n nodes, and let x be a node or an edge we wish to search for. If*

1. *For any edge e, info($\{e\}$) can be computed in time t_1.*
2. *For any neighboring clusters C_1 and C_2, info($C_1 \cup C_2$) can be computed in time t_2, given info(C_1) and info(C_2).*
3. *For any pair of neighboring clusters C_A, C_B such that $C_A \cup C_B = T$, we can decide if $x \in C_A$ or $x \in C_B$ in time t_3, given info(C_A) and info(C_B).*

Then we can maintain info for all trees in a dynamic forest in $O(t_1 + t_2 \log n)$ worst case time per link and cut and we can find x in time $O((t_2+t_3)\log n)$ worst case, given the ability to use $O((t_1+t_2)n)$ time and $O(n)$ space for preprocessing.

Proof: By theorem 1 we can maintain *info* for all trees in worst case time $O(t_1 + t_2 \log n)$ per *link* and *cut* as desired. For any cluster C and any node v, let $\overline{C}_v$ denote the subtree in $T \setminus C$ having v as its only boundary node.

Let $C = A \cup B$ be a cluster where x belongs to either A or B. Let $\partial A = \{a, c\}$, $\partial B = \{c, b\}$, then $\partial C \subseteq \{a, b\}$. We have three cases:

If $\partial C = \emptyset$ then C is the root of the topology tree and A and B are neighboring clusters with $A \cup B = T$. So we can in $O(t_3)$ time decide if $x \in A$ or $x \in B$. Assume w.l.o.g. that $x \in A$ then $info(\overline{A}_c) = info(B)$, and we continue the search in A.

If $|\partial C| = 1$, (assume w.l.o.g. that $\partial C = \{a\}$). Let $C_A = A \cup \overline{C}_a$. Then C_A and B are neighboring clusters, and $C_A \cup B = T$. If we have previously computed *info*$(\overline{C}_a)$ then we can in $O(t_2)$ time compute *info*(C_A). In $O(t_3)$ time we can then decide if $x \in C_A$ or $x \in B$. If $x \in A$ we can in $O(t_2)$ time compute *info*$(\overline{A}_a)$ and *info*$(\overline{A}_c)$ and continue the search in A. If $x \in B$ we have *info*$(\overline{B}_c)$ = *info*(C_A) and we may continue the search in B.

If $|\partial C| = 2$ then $\partial C = \{a, b\}$. Let $C_A = A \cup \overline{C}_a$ and $C_B = B \cup \overline{C}_b$. Then C_A and C_B are neighboring clusters, and $C_A \cup C_B = T$. If we have previously computed *info*$(\overline{C}_a)$ and *info*$(\overline{C}_b)$ we can in $O(t_2)$ time compute *info*(C_A) = *info*$(A \cup \overline{C}_a)$ and *info*(C_B) = *info*$(B \cup \overline{C}_b)$. In $O(t_3)$ time we can then decide if $x \in C_A$ or $x \in C_B$. W.l.o.g. $x \in C_A$ means that $x \in A$ since we knew $x \in C$. We then compute in $O(t_2)$ time *info*$(\overline{A}_a)$ and *info*$(\overline{A}_c)$ and continue the search in A.

Thus starting at the root of the topology tree the search uses $O(t_2 + t_3)$ time for each of the $O(\log n)$ levels, yielding a total time of $O((t_2 + t_3)\log n)$. □

4 Applications

In the following applications, we will use the following scheme: first we decide which information is sufficient to answer the question and next how to make that information available.

4.1 Dynamic 1-center

For any tree T and node v let $h_v(T)$ be the length of the longest path from v in T. The *1-center* problem is then finding a node v minimizing $h_v(T)$. For any node v let $p(v)$ be a node in T with maximal distance to v. It is well-known that for all v, *diam*(T) = *dist*$(p(v), p(p(v)))$ and thus *1-center*$(T) \subseteq p(v) \cdots p(p(v))$.

Lemma 3 *Let T be a tree, let $\mathcal{C}$ be a 1-center of T and let A and B be neighboring clusters with $A \cap B = \{c\}$ and $A \cup B = T$. Then $h_c(A) \geq h_c(B) \Rightarrow \mathcal{C} \in A$*

Proof: If $h_c(A) = h_c(B)$ then $\mathcal{C} = c$ and thus $\mathcal{C} \in A$ as stated. If $h_c(A) > h_c(B)$ then $h_c(A) = dist(c, p(c))$ and $p(c) \in A$. Now either $p(p(c)) \in A$ in which case $p(c) \cdots p(p(c)) \subseteq A$ and thus $\mathcal{C} \in A$ as stated, or $p(p(c)) \notin A$ in which case $c \in p(c) \cdots p(p(c))$, and $h_c(B) = dist(c, p(p(c)))$. Since $h_c(A) > h_c(B)$ we have $dist(c, p(c)) > dist(c, p(p(c)))$ and thus $\mathcal{C} \notin c \cdots p(p(c)) \setminus \{c\}$ hence $\mathcal{C} \in p(c) \cdots c \subseteq A$ as desired. □

For every cluster C, $\partial C = \{a, b\}$ we maintain:

- The distance between the boundary nodes: $dist(a, b)$
- The maximal distance in C from each boundary node: $h_a(C), h_b(C)$

Theorem 4 *The 1-center can be maintained dynamically under link, cut and change of edge weights in $O(\log n)$ worst case time per operation.*

Proof: For any edge e we can find $info(\{e\})$ in constant time. Furthermore, given two neighboring clusters C_1, C_2 and $info(C_1), info(C_2)$ we can find $info(C_1 \cup C_2)$ in constant time. Let A and B be neighboring clusters with $A \cap B = \{c\}$ and $A \cup B = T$. By lemma 3 we can in constant time decide whether the 1-center is located in the cluster A by testing if $h_c(A) \geq h_c(B)$. Thus by theorem 2 the 1-center can be maintained in $O(\log n)$ worst case time per *link* or *cut*. When an edge cost is changed we only need to update *info* for the $O(\log n)$ clusters containing it. Thus *change* can also be done in $O(\log\ n)$ worst case time. □

4.2 Dynamic 1-median

The *1-median* problem is finding a node $\mathcal{M}$ minimizing $\sum_{v \in V} weight(v) * dist(v, \mathcal{M})$, where $dist(v, \mathcal{M})$ is the sum of *cost* of edges on the unique path from v to $\mathcal{M}$ in the tree. For any tree T, let $w(T)$ denote the sum of node weights of T.

The lemma below follows from Goldman [9].

Lemma 5 *Let T be a tree, let $\mathcal{M}$ be a 1-median of T and let A and B be neighboring clusters with $A \cap B = \{c\}$ and $A \cup B = T$. Then $w(A) \geq w(B) \Rightarrow \mathcal{M} \in A$.* □

So given two neighboring subtrees whose union is T the *1-median* node is in the subtree with greatest weight. It follows that all we have to maintain for each cluster C is $info(C) = w(C \setminus \partial C)$

Theorem 6 *The 1-median can be maintained dynamically under link, cut and change of edge/node weights in $O(\log n)$ worst case time per operation.*

Proof: For any edge e we can find *info*($\{e\}$) in constant time. Furthermore, given two neighboring clusters C_1, C_2 and *info*(C_1), *info*(C_2) we can find *info*($C_1 \cup C_2$) in constant time. Let A and B be neighboring clusters with $A \cap B = \{c\}$ and $A \cup B = T$. By lemma 5 we can in constant time decide whether the 1-median is located in the cluster A by testing if $w(A \setminus \partial A) \geq w(B \setminus \partial B)$.

Thus by theorem 2 the 1-median can be maintained in $O(\log n)$ worst case time per *link* or *cut*. When a node weight is changed we only need to update *info* for the $O(\log n)$ clusters containing it as a non-boundary node. By lemma 5 an edge update does not alter which node is the *1-median*. Thus *change* can also be done in $O(\log\ n)$ worst case time. □

References

[1] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Minimizing diameters of dynamic trees. In *ICALP'97*, pages 270–280, 1997.

[2] V. Auletta, D. Parente, and G. Persiano. Dynamic and static algorithms for optimal placement of resources in a tree. *Theoretical Computer Science*, 165:441–461, 1996. See also ICALP'94.

[3] S. Cheng and M. Ng. Isomorphism testing and display of symmetries in dynamic trees. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'96)*.

[4] G.N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Computing*, 14(4):781–798, 1985.

[5] G.N. Frederickson. Parametric search and locating supply centers in trees. In *WADS'91*, volume 519, pages 299–319, 1991. see also SODA'91.

[6] G.N. Frederickson. Ambivalent data structures for dynamic 2–edge–connectivity and k smallest spanning trees. In *SIAM Journal on computing*, volume 26, pages 484–538, 1997. see also FOCS'91.

[7] G.N. Frederickson. A data structure for dynamically maintaining rooted trees. *Journal of Algorithms*, 24(1):37–65, 1997. See also SODA'93.

[8] B. Gavish and S. Sridhar. Computing the 2–median on tree networks in $O(n \log n)$ time. *Networks*, 26, 1995. see also Networks Vol. 27, 1996.

[9] A.J. Goldman. Optimal center location in simple networks. *Transportation Sci.*, 5:212–221, 1971.

[10] S.L. Hakimi and O. Kariv. An algorithmic approach to network location problems. ii: the p-medians. *SIAM J. APPL. MATH.*, 37(3):539–560, 1979.

[11] G.Y. Handler. Minimax location of a facility in an undirected tree network. *Transportation. Sci.*, 7:287–293, 1973.

[12] A. Rosenthal and J.A. Pino. A generalized algorithm for centrality problems on trees. *Journal of the ACM*, 36:349–361, 1989.

[13] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983. See also STOC'81.