

On the Longest Path Algorithm for Reconstructing Trees from Distance Matrices

Lev Reyzin, Nikhil Srivastava*

Department of Computer Science, Yale University, 51 Prospect St., New Haven, CT 06520, USA

Abstract

Culberson and Rudnicki [1] gave an algorithm that reconstructs a degree d restricted tree from its distance matrix. According to their analysis, it runs in time $O(dn \log_d n)$ for topological trees. However, this turns out to be false; we show that the algorithm takes $\Theta(n^{3/2}\sqrt{d})$ time in the topological case, giving tight examples.

Key words: Analysis of algorithms, graph algorithms

1 Introduction and Background

In [1], Culberson and Rudnicki consider the problem of reconstructing a degree d restricted weighted tree given its distance matrix D . If the tree has n vertices, D is an $n \times n$ matrix where each entry $d_{ij} \in \mathbb{R}_{\geq 0}$ gives the the weight of the (unique) path between vertices i and j . They describe an algorithm which finds the longest path in the tree, divides the remaining vertices into subtrees according to where they connect to this path, and then recurses on the subtrees. Their algorithm relies on three key ideas:

- (1) The longest path in a subtree can be computed in linear time. Simply pick an arbitrary vertex r and find the distances from r to every other vertex. Let u be the farthest vertex from r , and repeat to find the farthest vertex v from u . Then π_{uv} is the longest path in the tree, and we have looked at only two columns of D .

* Corresponding author.

Email addresses: lev.reyzin@yale.edu, nikhil.srivastava@yale.edu (Lev Reyzin, Nikhil Srivastava).

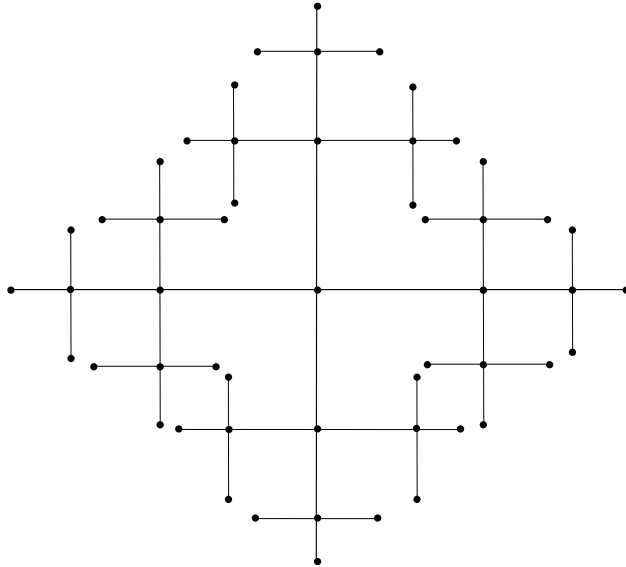


Fig. 1. Tree that minimizes the longest path in every subtree, from [1].

- (2) Given a longest path π_{uv} and distances computed in step (1), every other vertex z can be placed either on π_{uv} or on a subtree rooted at a known vertex w (a ‘hub’) on π_{uv} , with no additional queries to D . To be precise, z is in the subtree rooted at w iff $d_{zu} - d_{zv} = d_{wu} - d_{wv}$.
- (3) No further queries to D involving any hub w need be made, since for every vertex z in w ’s subtree, we have $d_{zw} = d_{zu} - d_{wu}$. This means that we can effectively forget about vertices that occur on the longest path, as far as queries to D are concerned.

The algorithm presented in [1] is equivalent to what is described above, with minor simplifications. We will call this algorithm LONGESTPATH.

In their analysis, Culberson and Rudnicki refer to the lookups to D in step (1) as ‘hub computations’ and establish that the running time is dominated by them. They claim that for topological trees (where all edge weights are 1), the running time of the algorithm is $O(dn \log_d n)$. Their claim rests on the following argument:

Since once a vertex...is located on a path in the tree it no longer participates in such computations in other partitions, the number of computations is maximized when the longest path in every subtree is minimized. When the maximum degree is restricted, this leads to balanced trees where all internal vertices are of maximum degree.

So according to [1], the worst case for degree four is the kind of tree shown in Figure 1.

Yet, this proof is incorrect. In section 2, we construct a topological tree on which the algorithm takes $\Omega(n^{3/2}\sqrt{d})$ time, contradicting the claim in [1]. In section 3 we present a revised analysis of LONGESTPATH, showing that $\Theta(n^{3/2}\sqrt{d})$ is tight for the topological case.

It is worth noting that other algorithms exist that achieve the $O(dn \log_d n)$ bound, which was also shown to be a lower bound in [6]. A quite different algorithm called DEEPESTPOINT¹, discovered by Hein [2], reconstructs both general and topological trees in $O(dn \log_d n)$ time. Further, variants of this problem have been studied in the experiment model for constructing evolutionary trees by Kannan et al. [3], Brodal et al. [4], and Lingas et al. [5]. Notwithstanding, LONGESTPATH is one of the first algorithms claimed to run in sub-quadratic time for tree reconstruction and also one of the simplest. This paper presents a correct analysis of it.

2 A Counterexample

While the tree in Figure 1 does minimize the lengths of the longest paths, it also ‘nicely’ splits the remaining vertices. Here, we take the opposite approach and consider a family of trees where removing the longest path always leaves the remaining vertices in a single subtree, as shown in Figure 2. Notice that G_i has $1 + 3 + \dots + (2i - 1) = i^2$ vertices and a longest path of length $2i - 1$ (perpendicular to the horizontal ‘stem’) – that is, the tree on n vertices has a longest path of length $2i - 1 = O(\sqrt{n})$ and a stem of length $i = O(\sqrt{n})$. Removing the longest path which is centered on the stem from G_i gives G_{i-1} .

For a tree of size n , finding a longest path lying vertically along the spine takes $\Omega(n)$ queries to D and leaves a subtree of size $\Omega(n - \sqrt{n})$. Hence, the total running time is at least

$$T(n) \geq n + T(n - \sqrt{n})$$

$$T(1) = 0.$$

The recurrence is solved easily by substitution:

$$\begin{aligned} T(n) &\geq n + (n - \sqrt{n}) + (n - \sqrt{n} - \sqrt{n - \sqrt{n}}) \dots \\ &\geq n + (n - \sqrt{n}) + (n - \sqrt{n} - \sqrt{n}) \dots \\ &= n + (n - \sqrt{n}) + (n - 2\sqrt{n}) \dots (n - \sqrt{n} \cdot \sqrt{n}) \\ &= \sqrt{n} \cdot n - \sqrt{n} \cdot \frac{\sqrt{n}(\sqrt{n} + 1)}{2} \end{aligned}$$

¹ In constructing DEEPESTPOINT[2], Hein mentions in passing that an algorithm recursively placing longest paths would run in $\Omega(n^{\frac{3}{2}})$

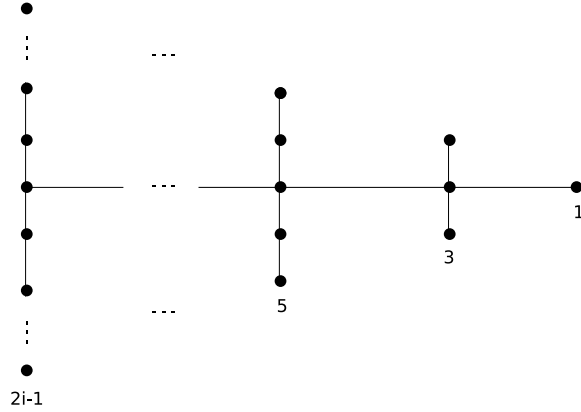


Fig. 2. A counterexample to the $O(dn \log_d n)$ analysis of [1], the tree G_i .

to get $T(n) = \Omega(n^{3/2})$.

All vertices in the above example are of degree at most 3. For the general degree d case, we consider trees G'_i that have $\frac{d}{2} - 1$ copies of each path on the stem. Observe that $|G'_i| = \Omega(d|G_i|)$ (since a linear number of vertices are copied $\Omega(d)$ times), so that longest paths in G'_i are of length $O(\sqrt{\frac{n}{d}})$. Since all longest paths of a given length intersect at only one vertex (which is on the stem), LONGESTPATH does not split them into smaller parts by placing them in separate subtrees. This gives the following recurrence:

$$T(n) \geq n + T\left(n - \sqrt{\frac{n}{d}}\right)$$

$$T(1) = 0$$

which has the solution $T(n) = \Omega(n^{3/2}\sqrt{d})$, showing that the $O(dn \log_d n)$ analysis of [1] is incorrect.

3 Analysis of the Topological Case

Say that a subtree is at *phase* j if it is obtained by recursing j times. Consider all subtrees $T_1 \dots T_k$ at a particular phase, with a total of n vertices. The number of queries to D used to find a longest path in T_i is just $O(|T_i|)$, so the total number of queries used in the phase is $\sum_i O(|T_i|) = O(n)$, i.e., *linear* in the total number of nodes. Thus we can bound the running time of LONGESTPATH by $n \times (\text{number of phases})$.

The correct analysis of LONGESTPATH relies on following observation:

Lemma 1 *Suppose π is a longest path in a subtree T , and S is the set of longest paths in T that are edge-disjoint with π . Then all paths in $S \cup \{\pi\}$ share a single vertex.*

PROOF. Suppose π_1 and π_2 are longest paths in T . Assume they do not intersect. Find the shortest path π' that connects π_1 and π_2 (we can do this since T is a tree). But now we can construct a path longer than π_1 : take the longer halves of π_1 and π_2 and join them with π' .

Thus every two longest paths in T intersect. Moreover, if π_1 and π_2 are edge-disjoint, they must intersect at a unique vertex: they cannot intersect at two consecutive vertices because then they would share an edge, and they cannot intersect at two non-consecutive vertices because that would imply a cycle in T .

Suppose $\pi' \in S$, and let π and π' intersect at v . Suppose $\pi'' \in S$ does not pass through v ; then, it must intersect π at a single vertex $w \neq v$. Also, π'' intersects π' at some vertex u not on π by the claim above. But now u, v, w lie on a cycle, which is impossible.

Lemma 2 *If the longest path at a phase has length l , then there are at most $O(dl)$ phases remaining before the algorithm terminates.*

PROOF. Suppose the longest path π chosen by LONGESTPATH has length l . Any longest paths that share an edge with π will be split into smaller paths, each of length at most $l - 1$, for the next phase. By lemma 1 all longest paths that are edge-disjoint from π must pass through a single vertex, which has degree at most d . Thus there are at most d such paths, and after d phases, the longest path remaining will be of length at most $l - 1$. Therefore, after dl phases, all longest paths are of length 0, and LONGESTPATH terminates.

Theorem 3 LONGESTPATH runs in $O(n^{3/2}\sqrt{d})$ time.

PROOF. If at any phase the longest path is of length at most $\sqrt{\frac{n}{d}}$, then by lemma 2 the number of phases remaining is $O(d\sqrt{\frac{n}{d}})$. Since each phase takes linear time, the total time starting from such a phase is $O(nd\sqrt{\frac{n}{d}}) = O(n^{3/2}\sqrt{d})$.

So assume the input has a longest path of length $l > \sqrt{\frac{n}{d}}$. How many phases can go by without l falling below $\sqrt{\frac{n}{d}}$? If $l \geq \sqrt{\frac{n}{d}}$ then at least $\sqrt{\frac{n}{d}}$ vertices are removed in each phase; thus, the number of such phases is at most $n/\sqrt{\frac{n}{d}} =$

\sqrt{dn} . Again, each phase takes linear time, so we reach a phase with $l \leq \sqrt{\frac{n}{d}}$ in time $O(n^{3/2}\sqrt{d})$, as desired.

Acknowledgements

We would like to thank Dana Angluin for suggesting the importance of Lemma 2 and for helpful discussions.

References

- [1] J. C. Culberson, P. Rudnicki, A fast algorithm for constructing trees from distance matrices, *Inf. Process. Lett.* 30 (4) (1989) 215–220.
- [2] J. J. Hein, An optimal algorithm to reconstruct trees from additive distance data, *Bulletin of Mathematical Biology* 51 (5) (1989) 597–603.
- [3] S. K. Kannan, E. L. Lawler, T. J. Warnow, Determining the evolutionary tree using experiments, *J. Algorithms* 21 (1) (1996) 26–50.
- [4] G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, A. Ostlin, The complexity of constructing evolutionary trees using experiments, in: *Proc. 28th International Colloquium on Automata, Languages, and Programming*, Vol. 2076 of *Lecture Notes in Computer Science*, 2001, pp. 140–151.
- [5] A. Lingas, H. Olsson, A. Ostlin, Efficient merging, construction, and maintenance of evolutionary trees, in: *ICAL '99: Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, Springer-Verlag, London, UK, 1999, pp. 544–553.
- [6] V. King, L. Zhang, Y. Zhou, On the complexity of distance-based evolutionary tree reconstruction, in: *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003, pp. 444–453.