A Near-Optimal Algorithm for Finding an Optimal Shortcut of a Tree*

Eunjin Oh[1] and Hee-Kap Ahn[2]

1 **Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea**
jin9082@postech.ac.kr

2 **Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea**
heekap@postech.ac.kr

Abstract

We consider the problem of finding a shortcut connecting two vertices of a graph that minimizes the diameter of the resulting graph. We present an $O(n^2 \log^3 n)$-time algorithm using linear space for the case that the input graph is a tree consisting of n vertices. Additionally, we present an $O(n^2 \log^3 n)$-time algorithm using linear space for a continuous version of this problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Network Augmentation, Shortcuts, Diameter, Trees

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.59

1 Introduction

Consider a graph $G = (V, E)$ with n vertices whose edges are assigned positive weights. The *length* of a path in G is the sum of the weights of the edges in the path. For any two vertices u and v in V, the *distance* between u and v in G is the length of the shortest path in G connecting u and v. We denote the distance between u and v by $d_G(u, v)$, or simply by $d(u, v)$ when it is understood in the context. The *diameter* of G is the maximum distance between any two vertices in G, that is, $\max_{u,v \in V} d_G(u, v)$.

In this paper, we consider *the diameter-optimal augmentation problem* for trees. We are to find a *shortcut* st that connects two vertices s and t of an input tree G and minimizes the diameter of $G + st = (V, E \cup \{st\})$. Here, the weight of a shortcut connecting two vertices is given in advance as an input. We assume that the weight of any shortcut can be determined in $O(1)$ time.

Related Work. For the case that the input graph is a path embedded in a metric space, Große et al. [6] gave an $O(n \log^3 n)$-time algorithm that computes an optimal shortcut by using parametric search [10]. Their decision algorithm uses the fact that for a fixed $s \in V$, the diameter of $T + st$ is the maximum of four monotone functions of vertex t moving along the path from s to an endpoint of the path.

Very recently, De Carufel et al. [4] studied the continuous version of this problem in which the input graph G is embedded in the Euclidean plane, the weight of an edge or a

* This work was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

shortcut is the Euclidean distance between its endpoints, and shortcuts are allowed to have their endpoints on any points of edges of G. Their goal is, by adding a shortcut to G, to minimize the *continuous diameter* of the resulting graph which is the maximum distance between any two *points* lying on edges and vertices. In this setting, they gave a linear-time algorithm for this problem when G is a path. They also studied the case that G is a cycle and showed that a single shortcut cannot decrease the continuous diameter. They gave a linear-time algorithm for computing an optimal pair of shortcuts when G is a convex cycle.

In graphs which are not necessarily embedded in a metric space, a similar problem was also studied. For the case that we are allowed to add more than one shortcut to an input graph, the problem of finding a diameter-optimal set of shortcuts becomes NP-hard [11]. For a path and a cycle, an upper and a lower bounds of the achievable diameter of the resulting graph were given by [2, 3]. For an outerplanar graph, Ishii [7] gave a constant-factor approximation algorithm for this problem.

Another variant of this problem is to find a shortcut that minimizes the dilation of a graph, and it was considered for graphs embedded in the Euclidean space [1, 5, 8] and in a metric space [9].

Our Results. We consider the problem of computing an optimal (discrete) shortcut of a tree and present an $O(n^2 \log^3 n)$-time algorithm for the problem. This problem is a generalization of the metric shortcut problem for paths [6] in the sense that the input graph is a tree and it is not necessarily embedded in a metric space. To achieve this running time, we traverse the tree in an Euler tour and efficiently compute the diameter of the tree with a shortcut using a data structure. As a byproduct, we present an $O(n \log n)$-time algorithm for computing the diameter of an edge-weighted graph containing exactly one cycle.

In addition, we consider its continuous version and present an algorithm for computing an optimal continuous shortcut when the input graph is a tree. Again the input graph is not necessarily embedded in a metric space, and therefore this problem is a generalization of the continuous Euclidean diameter problem [4]. Our algorithm takes $O(n^2 \log^3 n)$ time using $O(n)$ space.

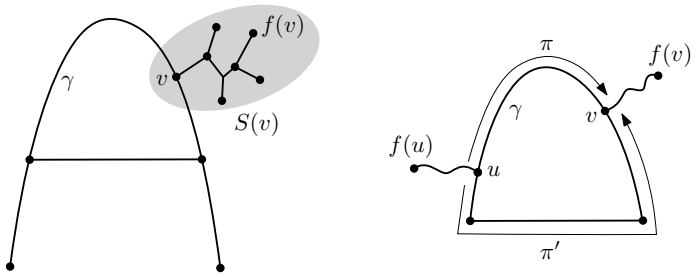
A Lower Bound for Computation. In the discrete version, consider a path with n vertices whose diameter is $\lambda \in \mathbb{R}$. There are at most $\binom{n}{2}$ shortcuts[1] in the input graph. Consider the case that only one shortcut has a sufficiently small weight $\epsilon > 0$ and the other shortcuts have the weight larger than λ. In this case, the shortcuts with larger weights do not decrease the diameter while the shortcut with weight ϵ decreases the diameter. Thus an optimal shortcut is the one with weight ϵ. However, there is no way to find the optimal shortcut unless we check the weights of all shortcuts. So it is inevitable to spend $\Omega(n^2)$ time to compute an optimal shortcut even in a path. A similar argument also holds in the continuous diameter problem.

Note that the algorithms in [4] and [6] assume that an input graph is embedded in a metric space. This allows them to achieve a near-linear or a linear running time.

2 Computing the Diameter of a Unicyclic Graph

We say a graph is *unicyclic* if the graph contains exactly one cycle. Consider a unicyclic graph G, where each edge e is assigned a positive weight $\ell(e)$. Let γ denote the unique cycle of G.

[1] Precisely speaking, there are $\binom{n}{2} - (n-1)$ shortcuts for a tree.



(a) (b)

Figure 1 (a) The gray region indicates the tree $S(v)$ rooted at v in $G^{\setminus\gamma}$. (b) For any two vertices u and v on γ, the clockwise vertex-weighted distance is $\delta_{cw}(u,v) = w(u) + \sum_{e\in\pi} \ell(e) + w(v)$ and the counterclockwise vertex-weighted distance is $\delta_{ccw}(u,v) = w(u) + \sum_{e\in\pi'} \ell(e) + w(v)$.

In this section, we present an $O(n \log n)$-time algorithm for computing the diameter of a unicyclic graph. Recall that the diameter of a graph is the maximum distance between any two vertices of the graph.

We can compute an optimal shortcut in $O(n^3 \log n)$ time by applying this algorithm for every possible shortcut. In Section 3.1 and 3.2, we give a faster algorithm for computing an optimal shortcut. Although the faster algorithm does not directly use the algorithm in this section, we first introduce this algorithm because the faster algorithm uses a technique similar to the one used in this algorithm.

2.1 Computing the Weights of Cycle Vertices

Consider the subgraph $G^{\setminus\gamma}$ of G obtained by removing the edges on γ. Each connected component of $G^{\setminus\gamma}$ forms a tree and contains exactly one vertex on γ. We treat each connected component as a tree rooted at the vertex on γ. For each vertex v on γ, we use $S(v)$ to denote the connected component rooted at v in $G^{\setminus\gamma}$. See Figure 1(a). Note that $S(v)$ may consist of only one vertex v. In addition, we use $f(v)$ to denote the vertex in $S(v)$ farthest from v. The vertex $f(v)$ for every vertex v on γ can be computed in $O(n)$ total time.

Now we annotate a weight $w(v)$ to each vertex v on γ, where $w(v) = d_G(v, f(v))$. If $S(v)$ consists of v alone, we set $d_G(v, f(v)) = 0$.

Vertex-Weighted Distances for Cycle Vertices. There are two simple paths connecting two vertices on γ. To make the description easier, we assign an arbitrary orientation to γ and treat it as a *clockwise orientation*. Then the opposite orientation is a *counterclockwise orientation*.

We define three *vertex-weighted* distances from u to v for any two distinct vertices u and v in γ as follows. (For an illustration, see Figure 1(b).)

1. The clockwise vertex-weighted distance : $\delta_{cw}(u,v) = w(u) + \sum_{e\in\pi} \ell(e) + w(v)$, where π is the simple path from u to v in the clockwise orientation along γ.
2. The counterclockwise vertex-weighted distance : $\delta_{ccw}(u,v) = w(u) + \sum_{e\in\pi'} \ell(e) + w(v)$, where π' is the simple path from u to v in the counterclockwise orientation along γ.
3. The vertex-weighted distance : $\delta(u,v) = \min\{\delta_{cw}(u,v), \delta_{ccw}(u,v)\}$.

For any vertex u in γ, let $\delta(u,u) = \delta_{cw}(u,u) = \delta_{ccw}(u,u) = 0$.

2.2 Computing the Diameter of a Unicyclic Graph

In this section we give an $O(n \log n)$-time algorithm to compute the diameter of G. A diametral pair of a graph is a pair (x, y) of vertices of the graph such that the distance between x and y is the same as the diameter of the graph. There are two possible cases for a diametral pair (x, y) of the unicyclic graph G:

1. Both x and y are contained in $S(v)$ for some vertex v on γ.
2. Vertex x is contained in $S(u)$ and vertex y is contained in $S(v)$ for two distinct vertices u and v on γ.

For the first case, a simple path connecting x and y is unique. And it is contained in $S(v)$. Therefore we can handle this case by computing the diameter and the diametral pair of $S(v)$. For all vertices v on γ, the diameter of $S(v)$ can be computed in total linear time.

For the second case, there are two simple paths connecting x and y, one through the clockwise path from u to v and one through the counterclockwise path from u to v, and we observe that $d_G(x, y) = \delta(u, v)$. Therefore, it suffices to find the *vertex-weighted diameter* of γ, that is, $\max_{u,v \in \gamma} \delta(u, v)$.

To handle the second case, we first compute the weight $w(v)$ of every vertex v in γ and annotate it to v, which takes $O(n)$ time in total for all vertices in γ. To compute the vertex-weighted diameter of γ, we find the maximum vertex-weighted distance $\lambda_v = \max_{u \in \gamma} \delta(v, u)$ for each vertex v in γ. By definition, the vertex-weighted diameter of γ is the maximum of λ_v over all vertices v in γ.

Computing λ_v for a vertex v in γ. Let $\lambda > 0$ be any real number. For a vertex v in γ, let $U_\lambda(v)$ be the set of all vertices u in γ with $\delta_{cw}(v, u) > \lambda$. Note that the vertices in $U_\lambda(v)$ are not necessarily consecutive along γ. Let $g_\lambda(v)$ be the vertex of γ with the largest counterclockwise distance $\delta_{ccw}(v, g_\lambda(v))$ among all vertices in $U_\lambda(v)$. If $U_\lambda(v) = \emptyset$, we let $g_\lambda(v)$ be v. Then the following observation holds as $\delta_{cw}(v, u) > \lambda$ for any $u \in U_\lambda(v)$.

▶ **Observation 1.** *We have $\delta_{ccw}(v, g_\lambda(v)) \leq \lambda$ if and only if $\lambda_v \leq \lambda$.*

To use this observation, we map each vertex u in γ to the point $p_v(u)$ with x-coordinate $\delta_{cw}(v, u)$ and y-coordinate $\delta_{ccw}(v, u)$ in the xy-plane. Let P_v be the set of points $p_v(u)$ for all vertices u in γ. For an illustration, see Figure 2(a). To check if $\lambda_v \leq \lambda$, it is sufficient to find the point with largest y-coordinate among all points of P_v lying to the right of the vertical line $x = \lambda$. By definition, the point is $p_v(g_\lambda(v))$ if it exists. Otherwise, $g_\lambda(v) = v$.

We use a 1-dimensional range tree on (the x-coordinates of) P_v to compute $g_\lambda(v)$ efficiently for any $\lambda > 0$. To be specific, we construct the range tree on P_v with respect to the x-coordinates of the points of P_v. Each node in the range tree corresponds to an interval on $\mathbb{R}$. For each node z, we store the largest y-coordinate of the points in P_v whose x-coordinate is in the interval corresponding to z. We denote the value stored at node z by y_z. Once we have this range tree, we can check in $O(\log n)$ time whether $\lambda_v \leq \lambda$.

In addition, we compute λ_v using the range tree on P_v. Starting from the root of the range tree, we traverse the range tree to some leaf. Suppose that we reach an internal node z in the range tree. The interval corresponding to z is subdivided into two subintervals corresponding to its two children. Let $\lambda \in \mathbb{R}$ be a value separating the two subintervals. We check whether $\lambda_v \leq \lambda$ or not in constant time by using the value, y_z, stored in z: $\lambda_v \leq \lambda$ if and only if $y_z \leq \lambda$. If $\lambda_v \leq \lambda$, we move to its left child. Otherwise, we move to its right child. In each node, we spend $O(1)$ time. Thus in $O(\log n)$ time, we can compute two consecutive points p_1 and p_2 in P_v such that $\lambda_1 \leq \lambda_v \leq \lambda_2$, where λ_1 and λ_2 are the x-coordinates of p_1 and p_2, respectively. See Figure 2(a).

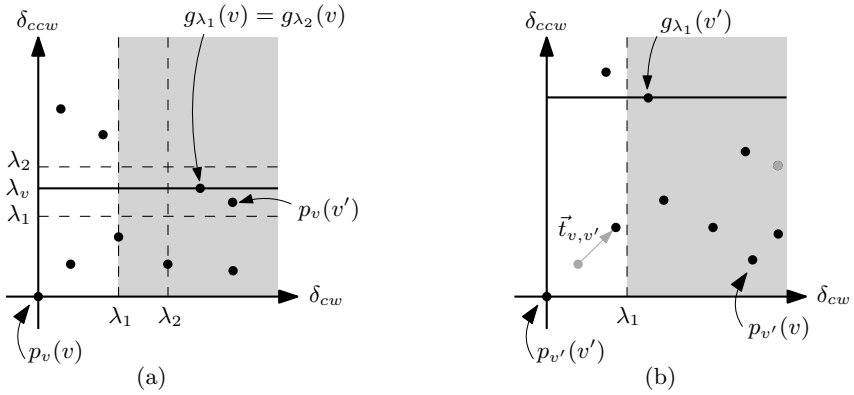


Figure 2 (a) The point set P_v obtained from the mapping of $\delta_{cw}(v,u)$ and $\delta_{ccw}(v,u)$ for each u in γ. We have $\lambda_1 < \lambda_v < \lambda_2$ because $\lambda_1 < \delta_{ccw}(v, g_{\lambda_1}(v))$ and $\delta_{ccw}(v, g_{\lambda_2}(v)) < \lambda_2$. Moreover, we have $\lambda_v = \delta_{ccw}(v, g_{\lambda_1}(v))$. (b) The point set $P_{v'}$ is a translated copy of P_v by $\vec{t}_{v,v'}$, with exceptions of $p_{v'}(v') \in P_{v'}$ and $p_v(v') \in P_v$.

Here, we have $g_{\lambda_1}(v) = g_{\lambda_2}(v)$. Moreover, λ_v is the y-coordinate of $g_{\lambda_1}(v)$. Therefore, we can compute λ_v in $O(\log n)$ time once we have the range tree on P_v.

▶ **Lemma 2.** *Once we have the 1-dimensional range tree on the x-coordinates of P_v for a vertex v in γ, the distance λ_v can be computed in $O(\log n)$ time.*

Computing λ_v for all vertices v in γ. Instead of computing the 1-dimensional range trees of P_v for all vertices v in γ repeatedly, we consider the vertices one by one in clockwise order along γ and make use of the 1-dimensional range tree on P_v in computing the 1-dimensional range tree on $P_{v'}$, where v' is the clockwise neighbor of v along γ.

Suppose that we have the 1-dimensional range tree on P_v. We show how to compute $P_{v'}$ using P_v for the clockwise neighbor v' of v. Consider a vertex $u \neq v, v'$ in γ. Recall that $p_v(u)$ is the point in the xy-plane with x-coordinate $\delta_{cw}(v,u)$ and y-coordinate $\delta_{ccw}(v,u)$. Let (x,y) be the x- and y-coordinates of $p_v(u)$. Then we have

$$p_{v'}(u) = (x - w(v) + w(v') - \ell(vv'),\ y - w(v) + w(v') + \ell(vv')).$$

Note that $w(v), w(v')$ and $\ell(vv')$ are independent to vertex u. This means that the point set $P_{v'} \setminus \{p_{v'}(v), p_{v'}(v')\}$ is a translated copy of the point set $P_v \setminus \{p_v(v), p_v(v')\}$ by the translation vector $\vec{t}_{v,v'} = (-w(v) + w(v') - \ell(vv'), -w(v) + w(v') + \ell(vv'))$. See Figure 2(b).

Thus, to compute $P_{v'}$, we remove $p_v(v')$ and $p_v(v)$ from P_v, translate the remaining points in P_v by $\vec{t}_{v,v'}$, and add $p_{v'}(v')$ and $p_{v'}(v)$ to the point set. The resulting point set is exactly $P_{v'}$. Both removing and adding the two points can be done in $O(\log n)$ time. The translation can be done in $O(1)$ time by moving the x-axis and the y-axis by $-\vec{t}_{v,v'}$, instead of translating the points.

The 1-dimensional range tree on $P_{v'}$ remains the same, except for the nodes for $p_{v'}(v'), p_{v'}(v)$ and the value each node stores. We first remove two points $p_{v'}(v'), p_{v'}(v)$ from the range tree on P_v. Then we update the values stored in the nodes of the range tree. These values increase or decrease by the same amount in the range tree of $P_{v'}$, and therefore we simply maintain a global value for the range tree, the *offset* of the values, instead of updating each value. We apply this offset to each value when it is used. After updating the offset, we add two points for v and v' to the 1-dimensional range tree in $O(\log n)$ time. The resulting tree is the range tree on $P_{v'}$.

Now, we have the 1-dimensional range tree on $P_{v'}$. By Lemma 2, we can compute $\lambda_{v'}$ in $O(\log n)$ time. By applying this procedure for all vertices one by one along γ, we have the following lemma.

▶ **Lemma 3.** *The distances λ_v for all vertices v in γ can be computed in $O(n \log n)$ time.*

Therefore, the following theorem holds.

▶ **Theorem 4.** *The diameter of a unicyclic graph G can be computed in $O(n \log n)$ time.*

3 The Diameter-Optimal Augmentation for a Tree

Let $T = (V, E)$ be an input tree, where each edge $e \in E$ is assigned a positive weight $\ell(e)$. In this section, we find a shortcut st connecting two vertices of T that minimizes the diameter of $T + st = (V, E \cup \{st\})$. We use λ^* to denote the diameter of $T + st$ for an optimal shortcut st. We assume that the weight of any shortcut can be determined in $O(1)$ time.

3.1 A Decision Algorithm for Computing a Shortcut

We first consider a decision problem and give an algorithm to decide whether $\lambda \geq \lambda^*$ or not in $O(n^2 \log n)$ time for a real number $\lambda > 0$. This decision algorithm is used as a subprocedure in the overall algorithm, which we will describe in Section 3.2.

Basically, we consider all possible shortcuts st and check whether the diameter of the unicyclic graph $T + st$ is at least λ. We spend $O(\log n)$ time for each shortcut.

3.1.1 The Euler Tour of the Input Tree

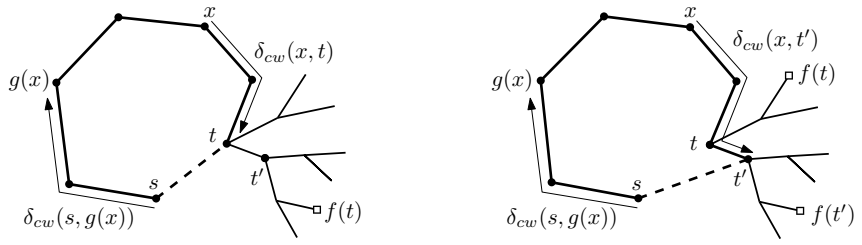
We fix a vertex s of T and consider it as an endpoint of a shortcut. In addition, we treat it as the root of T. An Euler tour of a graph is a path traversing each edge exactly once. For trees, we assume that each edge is bidirectional, so the Euler tour of a tree is the path through the tree that begins and ends at the root, traversing each edge exactly twice. In the Euler tour, we visit each vertex t of T exactly $\mathsf{deg}(t)$ times, where $\mathsf{deg}(t)$ is the degree of t in T.

Whenever we visit a vertex t in T, we compute the cycle of $T + st$ together with the weight $w(v)$ for each vertex v of the cycle, and check whether the diameter of $T + st$ is at least λ. Let $\gamma(s, t)$ be the cycle of $T + st$ for a vertex t in T.

When we move from a vertex t to one of its children t' in the Euler tour, the cycle changes locally. To be specific, assume that we already have $\gamma(s, t)$ and the weight $w(v)$ of each vertex v of $\gamma(s, t)$. As depicted in Figure 3, we construct $\gamma(s, t')$ by appending tt' and replacing st with st' to $\gamma(s, t)$. Then the weight of t may change and we need to compute $w(t')$ with respect to $\gamma(s, t')$, while the other vertices of $\gamma(s, t')$ have their weights unchanged. The weights, $w(t)$ and $w(t')$ with respect to $\gamma(s, t')$, can be computed in $O(1)$ time once we compute two distance values for every vertex in linear time in advance as follows.

For a vertex v of T rooted at s, let $h_1(v)$ be the distance between v and the descendant of v farthest from v. Let $h_2(v)$ denote the second largest value among values $h_1(u) + \ell(vu)$ for all children u of v. If v has exactly one child, we let $h_2(v) = 0$. We consider the vertices from the leaves one by one with respect to their depths, and compute $h_1(v)$ for all vertices v in total linear time. Then, we compute the distance $h_2(v)$ in $O(\deg(v))$ time by checking the values $h_1(u)$ for all children u of v.

The weight $w(t')$ with respect to $\gamma(s, t')$ is exactly $h_1(t')$, which can be found in constant time. For the weight $w(t)$ with respect to $\gamma(s, t')$, we have two cases: $w(t) = h_1(t)$ if



(a) (b)

Figure 3 When we move from vertex t to its neighbor vertex t', tt' is appended to the cycle $\gamma(s,t)$ and the weight $w(t) = d(t, f(t))$ of t with respect to the new cycle $\gamma(s,t')$ changes accordingly. (a) $h(x) = \delta_{cw}(x,t) + \delta_{cw}(s,g(x))$ with respect to $\gamma(s,t)$. (b) $h(x) = \delta_{cw}(x,t') + \delta_{cw}(s,g(x))$ with respect to $\gamma(s,t')$.

$h_1(t) > h_1(t') + \ell(tt')$, and $w(t) = h_2(t)$ otherwise. In either case, we can compute $w(t)$ in constant time.

When we move from a vertex t to its parent t'', we can obtain $\gamma(s,t'')$ and the weights of the vertices on $\gamma(s,t'')$ in constant time analogously.

▶ **Lemma 5.** *The cycle $\gamma(s,t')$ and the weights $w(v)$ of vertices v of $\gamma(s,t')$ can be updated in $O(1)$ time for a traversal of edge tt' in the Euler tour once we compute $h_1(v)$ and $h_2(v)$ for every vertex v of T in linear time.*

3.1.2 Deciding the diameter of $T + st$ for λ

Without loss of generality, we assume that t is the counterclockwise neighbor of s in $\gamma(s,t)$. We show how to decide whether the diameter of $T + st$ is at least λ in $O(\log n)$ time. To do this efficiently, we maintain a list of distance values sorted in the increasing order.

The Sorted List $\mathcal{H}$ of distance values. For each vertex x in $\gamma(s,t)$, let $g(x)$ denote the vertex with largest $\delta_{cw}(s,y)$ among vertices y lying in between s and x in clockwise direction from s and satisfying $\delta_{ccw}(x,y) > \lambda$. If no such vertex exists, we let $g(x)$ be *null*. Note that we have $\delta_{cw}(x,g(x)) = w(x) + d_T(x,t) + \ell(st) + d_T(s,g(x)) + w(g(x))$. Moreover, if $\delta_{cw}(x,g(x))$ is at most λ or $g(x)$ is null, then we have $\delta(x,y) \leq \lambda$ for every vertex y in between s and x in clockwise direction from s. Let $h(x) = w(x) + d_T(x,t) + d_T(s,g(x)) + w(g(x))$ for a vertex x in $\gamma(s,t)$ such that $g(x)$ is not null. We maintain $h(x)$ for all vertices x in $\gamma(s,t)$ such that $g(x)$ is not null and sort them in decreasing order. We denote this sorted list by $\mathcal{H}$.

Once we have the sorted list $\mathcal{H}$, we can check whether the diameter of $T + st$ is at least λ in constant time. We choose the first element of $\mathcal{H}$, say $h(x)$. Then the answer for the decision problem is "yes" if and only if $\ell(st) + h(x) \leq \lambda$. This can be checked in constant time.

Now we show how to update the sorted list $\mathcal{H}$ in $O(\log n)$ time when we move from a vertex t to a vertex t' in the Euler tour in three phases: (1) update $\mathcal{H}$, except for $h(t')$ and $h(t)$, (2) remove $h(t)$ from $\mathcal{H}$, and (3) compute $h(t')$ and $h(t)$, and insert them to $\mathcal{H}$. We compute $h(t)$ again because the value $h(t)$ changes. Let $\mathcal{H}$ be the sorted list which is already computed for the vertex t. We are to update $\mathcal{H}$, that is, to construct the sorted list for t'. Recall that in $\gamma(s,t')$, only one vertex is added to or removed from $\gamma(s,t)$. Here, we consider the case that one vertex, which is t', is added to $\gamma(s,t)$. The other case is analogous to this case.

First observe that $h(x)$ increases by the same amount for all vertices x in $\gamma(s,t') \setminus \{t,t'\}$. Moreover, this amount is exactly $\ell(tt') - w(t) + w'(t')$, where $w(t)$ and $w'(t')$ denote the weights of t and t' in $\gamma(s,t)$ and $\gamma(s,t')$, respectively. This means that the order for the elements in $\mathcal{H}$ remains the same, except for $h(t)$ and $h(t')$. Thus, we can update $\mathcal{H}$, except for $h(t')$ and $h(t)$, in constant time by maintaining the offset.

For computing $h(t)$, we observe that $g(t)$ remains the same. However, since the weight of t changes, the value $h(t)$ changes accordingly. With the new weight $w(t)$, we can update $h(t)$ in constant time and insert it to $\mathcal{H}$ in $O(\log n)$ time.

The remaining procedure is to compute $h(t')$. This procedure is similar to the procedure for computing the diameter of a unicyclic graph in Section 2. To this end, we maintain a point set and its range tree in addition to the sorted list $\mathcal{H}$.

The Point Set P and Its Range Tree. We map each vertex x in $\gamma(s,t')$ to the point $p(x) \in \mathbb{R}^2$ with $p(x) = (\delta_{ccw}(t',x), \delta_{cw}(s,x))$. Let P denote the set of $p(x)$'s for every vertex x in $\gamma(s,t')$. We construct a 1-dimensional range tree on P with respect to the x-coordinates of the points in the set. This range tree can be updated in $O(\log n)$ time as we did in Section 2.

Once we have the range tree, we can compute $h(t')$ in $O(\log n)$ time as follows. We find the point with largest y-coordinate in P among all points of P lying to the right of the vertical line $x = \lambda$. Note that the point with largest y-coordinate is $g(t')$ by definition. Thus, $h(t')$ is the y-coordinate of $p(g(t'))$ since $h(t') = w(t') + d_T(t',t') + d_T(s,g(t')) + w(g(t'))$. By adding $h(t')$ to its proper position in $\mathcal{H}$, we have the sorted list $\mathcal{H}$ for t'.

This completes the procedure for checking if the diameter of $T + st$ is at least λ, which takes $O(\log n)$ time. With these arguments, the following lemma holds.

▶ **Lemma 6.** *Given a tree T and a real number $\lambda > 0$, we can decide in $O(n^2 \log n)$ time whether $\lambda^* \leq \lambda$ or not.*

3.2 An Algorithm for Computing the Optimal Shortcut

The optimal diameter λ^* is either the distance $d(u,v)$ of two vertices $u, v \in T$ or the sum of two distances $d(u,t)$ and $d(s,v)$ of $u, v, s, t \in T$ and the weight of shortcut st. We first apply binary search on the lengths of all paths in T and find an interval of distance as described in Section 3.2.1. Then we consider the second case in Section 3.2.2.

3.2.1 Binary Search on Distances of Two Vertices

We can compute the set $\mathcal{D}_1$ of distances of every two vertices in T in $O(n^2)$ time and apply binary search on the distances using the decision algorithm in Section 3.1. This procedure gives us an interval containing no value of $\mathcal{D}_1$ in its interior but containing λ^* in $O(n^2 \log^2 n)$ time. However, this procedure requires $\Omega(n^2)$ space.

We show how to do this in $O(n^2 \log^3 n)$ time using $O(n)$ space. We apply binary search in $O(\log n)$ rounds. In the first round, we do the following: for each vertex v of T, we compute the median of the distances of v to all other vertices in T. This gives us n medians in $O(n^2)$ time using $O(n)$ space. Then we sort these medians and apply binary search on them. Let η be the interval obtained from the binary search. Note that η contains λ^*.

In the next rounds, we repeat the following and refine the interval containing λ^*. For each vertex v of T, we consider the distances of v to all other vertices of T that lie in the interval η and choose the median of them. Then we apply binary search again on these medians and reduce η into the subinterval that contains λ^*.

In every round, for each vertex v of T, the number of distances of $\mathcal{D}_1$ lying in η decreases by a constant fraction. Therefore, after $O(\log n)$ rounds, the interval η contains no distance of $\mathcal{D}_1$ in its interior. This takes $O(n^2 \log^3 n)$ time using $O(n)$ space.

3.2.2 Computing the Diameter with the Optimal Shortcut

Now we consider the case that λ^* is the sum of two distances $d(u,t)$ and $d(s,v)$ of $u,v,s,t \in T$ and the weight of shortcut st. We observe that u and v are contained in two different subtrees of $T^{\setminus\gamma(s,t)}$ rooted at x and y on $\gamma(s,t)$, respectively, such that $y = g(x)$. Therefore, we have $\lambda^* = h(x) + \ell(st)$. Based on this observation, we consider $O(n^2)$ candidate values one of which is exactly λ^* and find λ^* among them. We use a technique similar to parametric search [10].

We simulate the decision algorithm with λ^* without explicitly knowing λ^*. Let $\mathcal{D}_2$ be the set of candidate values that we consider. Initially, $\mathcal{D}_2$ is empty. We fix a vertex s in T and traverse the vertices in an Euler tour of T as we did in the decision algorithm. Assume that we move from t to the next vertex t'. We compute the cycle $\gamma(s,t')$ and the weights of vertices in $\gamma(s,t')$ in constant time once we have two distance values, $h_1(v)$ and $h_2(v)$, for every vertex v of T by Lemma 5.

Then we update the sorted list $\mathcal{H}$ in three phases. In the first phase, we update the elements in $\mathcal{H}$, except for $h(t')$ and $h(t)$. In the second phase, we remove $h(t')$ from $\mathcal{H}$. These two phases are independent of input λ, so the first phase can be done even though we do not have the explicit value of λ^*.

In the third phase, we first update the element for $h(t)$. This update is independent of λ because we already have $g(t)$, thus it can be done in $O(\log n)$ time. Then we update the element for $h(t')$. To do this, we maintain a point set and its range tree. Since these structures are independent of input λ, we can update them as we did in the decision algorithm without the explicit value of λ^*. To compute $h(t')$, we find the point with largest y-coordinate in P among all points lying to the right of $x = \lambda$. This procedure is dependent of an input λ.

However, we already have the interval η from Section 3.2.1 that contains λ^* but does not contain any value of $\mathcal{D}_1$ in its interior. Moreover, the x-coordinate of each point in P is the distance between two vertices of T. Therefore, we can find the point with largest y-coordinate in P among all points lying to the right of $x = \lambda^*$ by choosing any distance $\lambda \in \eta$. The third phase of the update can be done in $O(\log n)$ time.

The next step in the decision algorithm is to check whether $\lambda \geq \ell(st') + h(x)$ or not for the first element $h(x)$ of $\mathcal{H}$. This means that the value $\ell(st') + h(x)$ is a *critical* point in a sense that the answer for the decision problem with input λ depends on whether λ is at least this value or not. Thus, we add $\ell(st') + h(x)$ to $\mathcal{D}_2$.

We do this for all vertices s in T, and then we have $O(n^2)$ critical points one of which is exactly λ^*. The running time of this procedure is the same as the decision algorithm, which is $O(n^2 \log n)$. Then we apply binary search on $\mathcal{D}_2$ and we compute λ^*.

Since we have $O(n^2)$ distances, this requires $\Omega(n^2)$ space. Instead of computing the distances in $\mathcal{D}_2$ simultaneously, we apply binary search in $O(\log n)$ rounds as we did in Section 3.2.1. Then we can compute λ^* in $O(n^2 \log^3 n)$ time using $O(n)$ space.

▶ **Theorem 7.** *Given a tree T, an optimal shortcut can be computed in $O(n^2 \log^3 n)$ time using $O(n)$ space.*

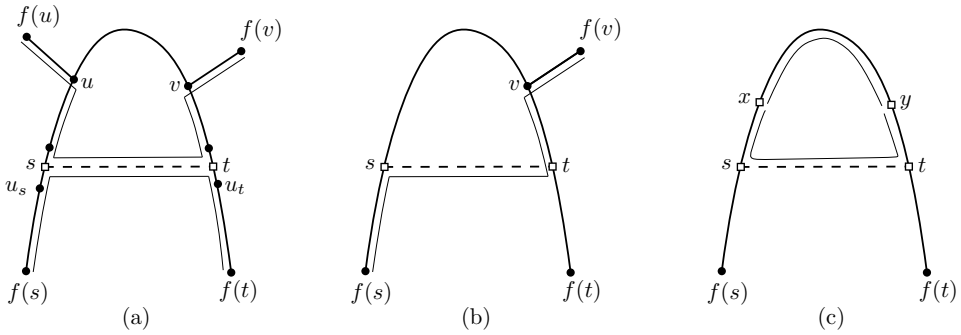


Figure 4 The points marked with disks are vertices of T while the points marked with squares are not necessarily vertices of T. (a), (b) Vertex-vertex pairs for a continuous diameter of a unicycle graph $T + st$. (c) A point-point pair (x, y).

4 The Continuous Diameter-Optimal Augmentation for a Tree

In this section, we consider a continuous version of the problem. We are given a tree $T = (V, E)$ with positive edge-weight $\ell(e)$ for each edge $e \in E$. In this problem, a subedge $e' \subset e$ also has its weights. For a point x in $e = ab$, the weight of the subedge ax is represented as an algebraic function with variable x, which is given as an input. Similarly, the weight of the subedge bx is given as an input. Since every edge has such two functions, we have $2n$ algebraic functions in total. With this weight, the length of the path between any two *points* of T is defined.

In addition, for two points $p \in e$ and $p' \in e'$, the weight of the shortcut pp' is also given as an input. We assume that for any two edges e and e', the weight of a shortcut pp' can be represented as an algebraic function with two variables $p \in e$ and $p' \in e'$. Since every pair of edges has such a function, we have at most $\binom{n}{2}$ algebraic functions representing the weight of a shortcut. In the following, we assume that we can compute the minimum of the upper envelope of any two algebraic functions with constant degree in constant time.

The continuous diameter is the maximum distance between any two points of T. Note that if (p, p') is a diametral pair for two points $p, p' \notin V$ then there are two simple paths connecting p and p' with the same length. The goal of this problem is to find two points s, t of T such that the continuous diameter of $T + st = (V, E \cup st)$ is minimized.

4.1 Characterization of the Continuous Optimal Shortcut

Let st be a continuous shortcut. Let $\gamma(s, t)$ be the unique cycle of $T + st$. For any vertex v on $\gamma(s, t)$, let $f(v)$ be the vertex of $S(v)$ farthest from v, where $S(v)$ is the connected component of $T^{\setminus\gamma(s,t)}$ containing v. Then there are two possible cases for a continuous diametral pair of the unicyclic graph $T + st$: (For an illustration, see Figure 4.)

- **vertex-vertex pair:** $(f(u), f(v))$ for two vertices $u, v \in \gamma(s, t)$ including s and t, and
- **point-point pair:** (x, y) for two points $x, y \in \gamma(s, t) \setminus V$.

For a point-point pair (x, y), there are two simple paths connecting x and y. Moreover, the two paths have the same length, which is half of the length of the cycle $\gamma(s, t)$.

4.2 A Decision Algorithm

We consider every pair (e_s, e_t) of edges of T and check whether there exists a shortcut st with $s \in e_s$ and $t \in e_t$ such that the diameter of $T + st$ is at most μ for a real number μ. If

such a shortcut exists, we say that (e_s, e_t) is *feasible*. We say that a shortcut st is *in* (e_s, e_t) if $s \in e_s$ and $t \in e_t$. To solve the decision problem efficiently, we fix an edge e_s and treat an endpoint of e_s as the root of the tree. Then we compute an Euler tour of T from the root and visit every edge twice along the tour. Whenever we visit an edge e_t, we check whether (e_s, e_t) is feasible.

Checking Whether an Edge Pair is Feasible. Let u_s and u_t be the endpoints of e_s and e_t, respectively, that do not lie on the cycle $\gamma(s,t)$ for some shortcut st in (e_s, e_t). Assume that u_s is the clockwise neighbor of u_t in $T + u_s u_t$. See Figure 4(a).

Let $f(v)$ be the vertex of $S(v)$ farthest from v, where $S(v)$ is the connected component of $T^{\setminus\gamma(u_s,u_t)}$ containing v for a vertex v on $\gamma(u_s, u_t)$. The weight $w(v)$ of v on the cycle is the distance between $f(v)$ and v.

Assume that we have the followings:

- $d_T(u_s, u_t)$, and
- for every vertex v in $\gamma(u_s, u_t)$, the value $h(v)$ which is the maximum $w(u) + d_T(u, u_s) + d_T(u_t, v) + w(v)$ over all vertices u lying in the path between u_s and v in clockwise order from v on T with $d_T(u, v) + w(v) + w(u) > \mu$.

Then we can check whether (e_s, e_t) is feasible or not in constant time as follows. Let $f(s,t)$ be the weight of the shortcut st in (e_s, e_t), which is an algebraic function with variables s and t. Let $f_s(s)$ be the weight of the subedge $u_s s$, which is an algebraic function. Similarly, let $f_t(t)$ be the weight of the subedge $u_t t$. The continuous diameter of the cycle $\gamma(s,t)$ is $(d_T(u_s, u_t) - f_s(s) - f_t(t) + f(s,t))/2$, which is half of the length of the cycle and the diameter is an algebraic function with variables s and t.

For a vertex-vertex diametral pair, we consider $h(v)$ value of a vertex v in $\gamma(u_s, u_t)$. We find the maximum of $h(v)$'s over all vertices v in $\gamma(u_s, u_t)$. Once $h(v)$'s are sorted in decreasing order, we can choose the maximum h in constant time. If $h - f_s(s) - f_t(t) + f(s,t) \leq \mu$ for some shortcut st in (e_s, e_t), then the diameter is at most μ if a diametral pair is a vertex-vertex pair.

Thus, we check whether there is a shortcut st such that the maximum of $(d_T(u_s, u_t) - f_s(s) - f_t(t) + f(s,t))/2$ and $h - f_s(s) - f_t(t) + f(s,t)$ is at most μ. If so, we conclude that st is feasible. Otherwise, st is not feasible. With the argument in this section, we have the following lemma.

▶ **Lemma 8.** *Once we have $d_T(u_s, u_t)$ and $h(v)$ for every vertex v in $\gamma(u_s, u_t)$, we can check whether (e_s, e_t) is feasible or not in constant time.*

The distance $d_T(u_s, u_t)$ can be updated in constant time if we consider the edge e_t along the Euler tour. For the values $h(v)$'s, we can use the algorithm in Section 3.1. The algorithm in Section 3.1 computes exactly these values by maintaining a point set and a range tree. Therefore, we have the following.

▶ **Lemma 9.** *Given a tree T and a real number $\mu > 0$, we can decide in $O(n^2 \log n)$ time whether μ is at most the optimal solution or not.*

4.3 An Overall Algorithm

Let $\mathcal{D}_1$ be the set of distances between every pair of vertices in T. As we did in the algorithm for the discrete version, we compute the interval μ containing the optimal solution μ^* but containing no value of $\mathcal{D}_1$ in $O(n^2 \log^3 n)$ time.

Then we simulate the decision algorithm in Section 4.2 with an optimal solution. This can be done as follows without explicitly knowing an optimal solution. Since we have the interval μ, we can apply the procedures in Section 4.2, except for the last one that compares the maximum of the two algebraic functions with an input. Instead of applying the last procedure, we compute the maximum and put it into the set $\mathcal{D}_2$, which is set to be empty initially. Then one of the values in $\mathcal{D}_1 \cup \mathcal{D}_2$ is the optimal solution. Thus, we again apply binary search on the set $\mathcal{D}_1 \cup \mathcal{D}_2$. This is similar to the algorithm for the discrete version. This algorithm can also be analyzed analogously. Thus, we have the following theorem.

▶ **Theorem 10.** *Given a tree T, the continuous optimal shortcut can be computed in $O(n^2 \log^3 n)$ time using $O(n)$ space.*

References

1 Hee-Kap Ahn, Mohammad Farshi, Christian Knauser, Michiel Smid, and Yajun Wang. Dilation-optimal edge deletion in polygonal cycles. *International Journal of Computational Geometry & Applications*, 20(1):69–87, 2010.

2 Noga Alon, Andras Gyarfas, and Miklos Ruszinko. Decreasing the diameter of bounded degree graphs. *Journal of Graph Theory*, 35(3):161–172, 2000.

3 F. R. K. Chung and M. R. Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8(4):511–534, 1984.

4 Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, and Michiel Smid. Minimizing the continuous diameter when augmenting paths and cycles with shortcuts. In *Proceedings of 15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, pages 27:1–27:14, 2016.

5 Mohammad Farshi, Panos Giannopoulos, and Joachim Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal on Computing*, 38(1):226–240, 2008.

6 Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel Smid, and Fabian Stehn. Fast algorithms for diameter-optimally augmenting paths. In *Proceedings of Automata, Languages, and Programming: 42nd International Colloquium (ICALP 2015)*, pages 678–688, 2015.

7 Toshimasa Ishii. Diameter bounds for altered graphs. *Journal of Graph Theory*, 74(4):392–416, 2013.

8 Rolf Klein, Christian Knauer, Giri Narasimhan, and Michiel Smid. Exact and approximation algorithms for computing the dilation spectrum of paths, trees, and cycles. In *Proceedings of the 16th International Symposium on Algorithms and Computatiom, (ISAAC 2005)*, pages 849–858, 2005.

9 Jun Luo and Christian Wulff-Nilsen. Computing best and worst shortcuts of graphs embedded in metric spaces. In *Proceedings of the 19th International Symposium on Algorithms and Computatiom, (ISAAC 2008)*, pages 764–775, 2008.

10 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

11 Anneke A. Schoone, Hans L. Bodlaender, and Jan van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11(3):409–427, 1987.