Almost Optimal Algorithms for Diameter-Optimally Augmenting Trees

Davide Bilò
Department of Humanities and Social Sciences, University of Sassari
Via Roma 151, 07100 Sassari (SS), Italy
davide.bilo@uniss.it
https://orcid.org/0000-0003-3169-4300

Abstract

We consider the problem of augmenting an n-vertex tree with one shortcut in order to minimize the diameter of the resulting graph. The tree is embedded in an unknown space and we have access to an oracle that, when queried on a pair of vertices u and v, reports the weight of the shortcut (u, v) in constant time. Previously, the problem was solved in $O(n^2 \log^3 n)$ time for general weights [Oh and Ahn, ISAAC 2016], in $O(n^2 \log n)$ time for trees embedded in a metric space [Große et al., `arXiv:1607.05547`], and in $O(n \log n)$ time for paths embedded in a metric space [Wang, WADS 2017]. Furthermore, a $(1 + \varepsilon)$-approximation algorithm running in $O(n + 1/\varepsilon^3)$ has been designed for paths embedded in $\mathbb{R}^d$, for constant values of d [Große et al., ICALP 2015].

The contribution of this paper is twofold: we address the problem for trees (not only paths) and we also improve upon all known results. More precisely, we design a *time-optimal* $O(n^2)$ time algorithm for general weights. Moreover, for trees embedded in a metric space, we design (i) an exact $O(n \log n)$ time algorithm and (ii) a $(1 + \varepsilon)$-approximation algorithm that runs in $O\big(n + \varepsilon^{-1} \log \varepsilon^{-1}\big)$ time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis, Theory of computation → Approximation algorithms analysis

Keywords and phrases Graph diameter, augmentation problem, trees, time-efficient algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.40

Related Version A full version of this paper can be found at `https://arxiv.org/abs/1809.08822`.

1 Introduction

Consider a tree $T = (V(T), E(T))$ of n vertices, with a *weight* $\delta(u, v) > 0$ associated with each edge $(u, v) \in E(T)$, and let $c : V(T)^2 \to \mathbb{R}_{\geq 0}$ be an unknown function that assigns a weight to each possible *shortcut* (u, v) we could add to T. For a given path P of an edge-weighted graph G, the *length* of P is given by the overall sum of its edge weights. We denote by $d_G(u, v)$ the *distance* between u and v in G, i.e., the length of a shortest path between u and v in G.[1] The *diameter* of G is the maximum distance between any two vertices in G, that is $\max_{u,v \in V(G)} d_G(u, v)$.

In this paper we consider the *Diameter-Optimal Augmentation Problem* (DOAP for short). More precisely, we are given an edge-weighted tree T and we want to find a shortcut (u, v) whose addition to T minimizes the diameter of the resulting (multi)graph, that we denote

[1] If u and v are in two different connected components of G, then $d_G(u, v) = \infty$.

by $T + (u, v)$. We assume to have (unlimited access to) an oracle that is able to report the weight of a queried shortcut in $O(1)$ time.

DOAP has already been studied before and the best known results are the following:

- an $O(n^2 \log^3 n)$ time and $O(n)$ space algorithm and a lower bound of $\Omega(n^2)$ on the time complexity of any exact algorithm [16];
- an $O(n^2 \log n)$ time algorithm for trees *embedded* in a metric space [11];
- an $O(n \log n)$ time algorithm for paths embedded in a metric space [18];[2]
- a $(1 + \varepsilon)$-approximation algorithm that solves the problem in $O(n + 1/\varepsilon^3)$ for paths embedded in the Euclidean (constant) k-dimensional space [10].

In this paper we improve upon (almost) all these results. More precisely:

- we design an $O(n^2)$ time and space algorithm that solves DOAP. We observe that the time complexity of our algorithm is optimal;
- we develop an $O(n \log n)$ time and $O(n)$ space algorithm that solves DOAP for trees embedded in a metric space;
- we provide a $(1 + \varepsilon)$-approximation algorithm, running in $O\left(n + \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)$ time and using $O(n + 1/\varepsilon)$ space, that solves DOAP for trees embedded in a metric space.

Our approaches are similar in spirit to the ones already used in [10, 11, 18], but we need many new key observations and novel algorithmic techniques to extend the results to trees. Our results leave open the problem of solving DOAP in $O(n^2)$ time and truly subquadratic space for general instances, and in $o(n \log n)$ time for trees embedded in a metric space.

Other related work. The variant of DOAP in which we want to minimize the *continuous diameter*, i.e., the diameter measured with respect to all the points of a tree (not only its vertices), has been also addressed. Oh and Ahn [16] designed an $O(n^2 \log^3 n)$ time and $O(n)$ space algorithm. De Carufel et al. [3] designed an $O(n)$ time algorithm for paths embedded in the Euclidead plane. Subsequently, De Carufel et al. [4] extended the results to trees embedded in the Euclidean plane by designing an $O(n \log n)$ time algorithm.

Several generalizations of DOAP in which the graph (not necessarily a tree) can be augmented with the addition of k edges have also been studied. In the more general setting, the problem is NP-hard [17], not approximable within logarithmic factors [2], and some of its variants – parameterized w.r.t. the overall cost of added shortcuts and resulting diameter – are even W[2]-hard [8, 9]. Therefore, several approximation algorithms have been developed for all these variations [2, 5, 7, 8, 14]. Finally, upper and lower bounds on the values of the diameters of the augmented graphs have also been investigated in [1, 6, 13].

Our approaches. Große et al. [10] were the first to attack DOAP for paths embedded in a metric space. They gave an $O(n \log n)$ time algorithm for the corresponding *search version* of the problem:

> *For a given value $\lambda > 0$, either compute a shortcut whose addition to the path induces a graph of diameter at most λ, or return $\perp$ if such a shortcut does not exist.*

Then, by implementing their algorithm also in a parallel fashion and applying Megiddo's parametric-search paradigm [15], they solved DOAP for paths embedded in a metric space in $O(n \log^3 n)$ time. Lately, Wang [18] improved upon this result in two ways. First, he solved the search version of the problem in linear time. Second, he developed an ad-hoc

[2] More precisely, c is a metric function and $\delta(u, v) = c(u, v)$, for every $(u, v) \in E(G)$.

algorithm that, using the algorithm for the search version of the problem black-box together with sorted-matrix searching techniques and range-minima data structure, is able to: (i) reduce the size of the solution-search-space from $\binom{n}{2}$ to n in $O(n \log n)$ and (ii) evaluate the quality of all the leftover solutions in $O(n)$ time.

Our approach for Doap instances embedded in a metric space is close in spirit to the approach used by Wang. In fact, we develop an algorithm that solves the search version of Doap in linear time and we use such an algorithm black-box to solve Doap in $O(n \log n)$ time and linear space by first reducing the size of the solution-search-space from $\binom{n}{2}$ to n and then by evaluating the quality of the leftover solutions in $O(n \log n)$ time. However, differently from Wang's approach, we use Hershberger data structure for computing the *upper envelope* of a set of linear functions [12] rather than a range-minima data structure. Furthermore, there are several issues we have to deal with due to the much more complex topology of trees. We solve some of these issues using a lemma proved in [11] about the existence of an optimal shortcut whose endvertices both belong to a diametral path of the tree. This allows us to reduce our Doap instance to a node-weighted path instance of a similar problem, that we call WDoap, in which the distance between two vertices is measured by adding the weights of the two considered vertices to the length of a shortest path between them, and the diameter is defined accordingly. However, it is not possible to use the algorithms presented in [10, 18] black-box to solve WDoap. Therefore we need to design an ad-hoc algorithm whose correctness strongly relies on the structural properties of diametral paths and properties satisfied by node weights. Furthermore, most of the easy observations that can be done for paths become non-trivial lemmas that need formal proofs for trees.

Our time-optimal algorithm that solves Doap for instances with general weights is based on the following important observations. We reduce, in $O(n^2)$ time, a Doap instance to another Doap instance in which the function c is *graph-metric*, i.e., c is an almost metric function that satisfies a weaker version of the triangle inequality. Since our $O(n \log n)$ time algorithm for Doap instances embedded in a metric space also works for graph-metric spaces, we can use this algorithm black-box to solve the reduced Doap instance in $O(n \log n)$ time, thus solving the original Doap instance in $O(n^2)$ time.

Finally, the $(1+\varepsilon)$-approximation algorithm for trees embedded in a metric space is obtained by proving that the diameter of the tree is at most three times the diameter, say D^*, of an optimal solution. This allows us to partition the vertices along a diametral path into $O(1/\varepsilon)$ sets such that the distance between any two vertices of the same set is at most $O(\varepsilon D^*)$. We choose a suitable *representative* vertex for each of the $O(1/\varepsilon)$ sets and use our $O(n \log n)$ time algorithm to find an optimal shortcut in the corresponding WDoap instance restricted to the set of representative vertices. Since the representative vertices are $O(1/\varepsilon)$, the optimal shortcut in the restricted WDoap instance can be found in $O(\varepsilon^{-1} \log \varepsilon^{-1})$ time. Furthermore, because of the choice of the representative vertex, we can show that the shortcut returned is a $(1+\varepsilon)$-approximate solution for the (unrestricted) WDoap instance of our problem, i.e., a $(1+\varepsilon)$-approximate solution for our original Doap instance.

Due to the lack of space, in this paper we only describe the $O(n \log n)$ time algorithm for the instances embedded in a metric space and we refer to `https://arxiv.org/abs/1809.08822` for the full version of the paper. The rest of the paper is organized as follows: in Section 2 we present some preliminary results among which the reduction from general instances to graph-metric instances; in Section 3 we describe the reduction from Doap to WDoap together with further simplifications; in Section 4 we design an algorithm that solves a search version of WDoap in linear time; in Section 5 we develop an algorithm that solves Doap for trees embedded in a graph-metric space.

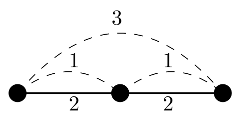

Figure 1 An example of a graph-metric function. The graph (a path in this specific example) is given by the two solid edges of weight 2 each. The shortcuts are dashed. The example shows a graph-metric function that does not satisfy the triangle inequality.

2 Preliminaries

To simplify the notation, we drop the subscript from $d_T(\cdot,\cdot)$ whenever T is clear from the contest and we denote $d_{T+(u,v)}(\cdot,\cdot)$ by $d_{u,v}(\cdot,\cdot)$. The diameter of a graph G is denoted by $\mathtt{diam}(G)$. A *diametral path* of G is a shortest path in G of length equal to $\mathtt{diam}(G)$. We say that c is a *graph-metric w.r.t. G*, or simply a *graph-metric* when G is clear from the contest, if, for every three distinct vertices u, v, and z of G, we have that

$$c(u,v) \le c(u,z) + d(z,v). \qquad (\textit{graph-triangle inequality})$$

We observe that a metric cost function is also graph-metric, but the opposite does not hold in general (see Figure 1). The *graph-metric closure* induced by c is a function $\bar{c}$ such that, for every two vertices u and v of G, $\bar{c}(u,v) = \min\{d_G(u,u')+c(u',v')+d_G(v',v) \mid u',v' \in V(G)\}$. The following lemma shows that we can restrict DOAP to input instances where c is graph-metric. We observe that the reduction holds for any graph and not only for trees.

▶ **Lemma 1.** *Solving the instance $\langle G,\delta,c\rangle$ of* DOAP *is equivalent to solving the instance $\langle G,\delta,\bar{c}\rangle$ of* DOAP*, where $\bar{c}$ is the graph-metric closure induced by c.*

Next lemma shows the existence of an optimal shortcut whose endvertices are both on a diametral path of T for the case in which c is a graph-metric.

▶ **Lemma 2.** *Let $\langle T,\delta,c\rangle$ be an instance of* DOAP*, where c is a graph-metric, and let $P = (v_1,\dots,v_N)$ be a diametral path of T. There always exists an optimal shortcut (u^*,v^*) such that $u^*, v^* \in V(P)$.*

3 Reduction from trees to node-weighted paths

In this section we show that a DOAP instance embedded in a graph-metric space can be reduced in linear time to a node-weighted instance of a similar problem. The *Node-Weighted-Diameter-Optimal Augmentation Problem* (WDOAP for short) is defined as follows:

Input: A path $P = (v_1,\dots,v_N)$, with a weight $\delta(v_i,v_{i+1}) > 0$ associated with each edge (v_i,v_{i+1}) of P, a weight $w(v_i)$ associated with each vertex v_i such that $0 \le w(v_i) \le \min\{d(v_1,v_i), d(v_i,v_N)\}$, and an oracle that is able to report the weight $c(v_i,v_j)$ of a queried shortcut in $O(1)$ time, where c is a graph-metric;

Output: Two indices i^* and j^*, with $1 \le i^* < j^* \le N$, that minimize the function

$$D(i,j) := \max_{1\le k<h\le N} \{w(v_k) + d_{v_i,v_j}(v_k,v_h) + w(v_h)\}.$$

We observe that $w(v_1) = w(v_N) = 0$. Let $\langle T,\delta,c\rangle$ be a DOAP instance, where c is a graph-metric. Let $P = (v_1,\dots,v_N)$ be a diametral path of T, T_i the tree containing v_i in the forest obtained by removing the edges of P from T, and $w(v_i) := \max_{v\in V(T_i)} d(v_i,v)$. We say that $\langle P,\delta,w,c\rangle$ is the WDOAP instance induced by $\langle T,\delta,c\rangle$ and P. The following lemma holds.

▶ **Lemma 3.** *The* WDOAP *instance* $\langle P, \delta, w, c\rangle$ *induced by* $\langle T, \delta, c\rangle$ *and* P *can be computed in* $O(n)$ *time. Moreover,* $\texttt{diam}\big(T + (v_i, v_j)\big) = D(i,j)$*, for every* $1 \leq i < j \leq N$*.*

3.1 Further simplifications

In the rest of the paper, we show how to solve WDOAP in $O(N \log N)$ time and linear space. To avoid heavy notation, from now on we denote a vertex v_i by using its associated index i. All the lemmas contained in this subsection are non-trivial generalizations of observations made in [10] for paths. We start proving a useful lemma.

▶ **Lemma 4.** *Let* i, j *be two indices such that* $1 \leq i < j \leq N$*. Let* $I = \{1\} \cup \{k \mid i < k \leq N\}$ *and let* $J = \{N\} \cup \{h \mid 1 \leq h < j\}$*. We have that*

$$D(i,j) = \max_{k \in I, h \in J, k<h} \big\{w(k) + d_{i,j}(k,h) + w(h)\big\}.$$

As we will see in a short, Lemma 4 allows us to decompose the function $D(i,j)$ into four monotone parts. First of all, for every $i = 1, \ldots, N$, we define

$$\omega(i) := \max\big\{w(j) - d(i,j) \mid 1 \leq j \leq N\big\}.$$

Observe that, for every $1 \leq i \leq j \leq N$,

$$\omega(i) \leq \omega(j) + d(i,j). \qquad \textit{(node-triangle inequality)}$$

Furthermore, $\omega(i) \geq w(i)$, for every $1 \leq i \leq N$, which implies $\omega(1) = \omega(N) = 0$. The following lemma establishes the time complexity needed to compute all the values $\omega(i)$.

▶ **Lemma 5.** *All the values* $\omega(i)$*, with* $1 \leq i \leq N$*, can be computed in* $O(N)$ *time.*

For the rest of this section, unless stated otherwise, i and j are such that $1 \leq i < j \leq N$. The four functions used to decompose $D(i,j)$ are the following (see also Figure 2)

$$U(i,j) := d(1,i) + c(i,j) + d(j,N);$$

$$S(i,j) := \max_{i \leq h < j} \Big(\omega(h) + \min\big\{d(1,h), d(1,i) + c(i,j) + d(h,j)\big\}\Big);$$

$$E(i,j) := \max_{i < k \leq j} \Big(\omega(k) + \min\big\{d(k,N), d(j,N) + c(i,j) + d(i,k)\big\}\Big);$$

$$C(i,j) := \max_{i < k < h < j} \Big(\omega(k) + \min\big\{d(k,h), d(i,k) + c(i,j) + d(h,j)\big\} + \omega(h)\Big).$$

Using both the graph-triangle inequality and the node-triangle inequality, we can observe that all the four functions are monotonic. More precisely:

- $U(i,j+1) \leq U(i,j) \leq U(i+1,j)$;
- $S(i-1,j) \leq S(i,j) \leq S(i,j+1)$;
- $E(i,j+1) \leq E(i,j) \leq E(i-1,j)$;
- $C(i+1,j) \leq C(i,j) \leq C(i,j+1)$.

Moreover, we can prove the following lemma.

▶ **Lemma 6.** $D(i,j) = \max\big\{U(i,j), S(i,j), E(i,j), C(i,j)\big\}$.

The following lemma allows us to efficiently compute the values $U(i,j), S(i,j)$, and $E(i,j)$.

▶ **Lemma 7.** *After a* $O(N)$*-time precomputation phase, for every* $1 \leq i < j \leq N$*,* $U(i,j)$ *can be computed in* $O(1)$ *time, while both* $S(i,j)$ *and* $E(i,j)$ *can be computed in* $O(\log N)$ *time.*

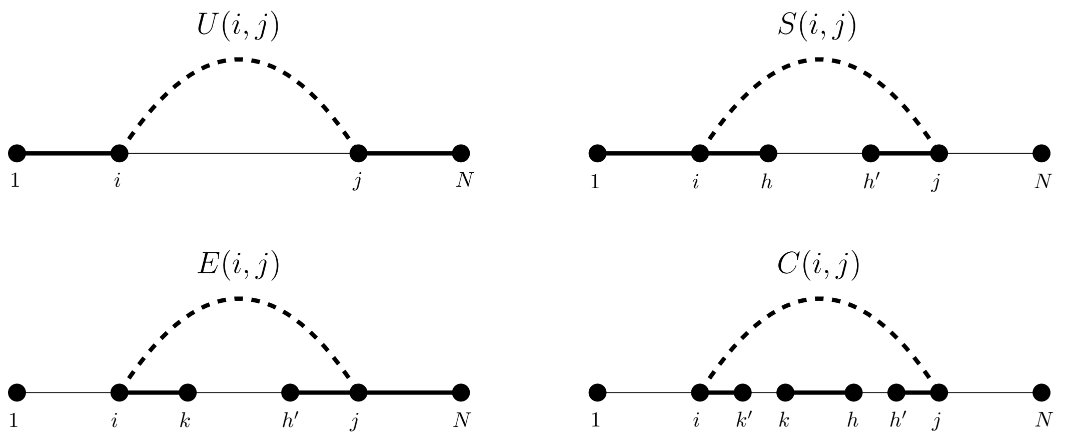

Figure 2 The four functions used to decompose $D(i,j)$. Node weights are omitted and shortest paths are highlighted in bold. $U(i,j) = d(1,i) + c(i,j) + d(j,N)$. $S(i,j)$ is the maximum among all the (node-weighed) distances between 1 and all the vertices of the cycle. In our example the distance from 1 to h is $d(1,h) + \omega(h)$, while the distance from 1 to h' is $d(1,i) + c(i,j) + d(j,h') + \omega(h')$. $E(i,j)$ is the maximum among all the distances between N and all the vertices of the cycle. In our example the distance from N to k is $d(k,N) + \omega(k)$, while the distance from N to k' is $d(j,N) + c(i,j) + d(i,k') + \omega(k')$. Finally, $C(i,j)$ is the maximum among all the distances between pair of vertices in the cycle. In our example the distance from k to h is $\omega(k) + d(k,h) + \omega(h)$, while the distance from k' to h' is $\omega(k') + d(i,k) + c(i,j) + d(j,h') + \omega(h')$.

4 The linear time algorithm for the search version of WDOAP

In this section we design an $O(N)$ time algorithm for the following search version of WDOAP:

> *For a given* WDOAP *instance* $\langle P, \delta, \omega, c\rangle$, *where* c *is a graph-metric and* ω *satisfies the node-triangle inequality, and a real value* $\lambda > 0$, *either find two indices* $1 \le i < j \le N$ *such that* $D(i,j) \le \lambda$, *or return* $\perp$ *if such two indices do not exist.*

In the following we assume that $d(1,N) > \lambda$, as otherwise $D(i,j) \le \lambda$ for any two indices i and j. For the rest of this section, unless stated otherwise, i and j are two fixed indices such that $1 \le i < j \le N$. Let $i < \mu_i \le N$ be the minimum index, or $N+1$ if such an index does not exists, such that $U(i,\mu_i) \le \lambda$. Our algorithm computes the index μ_i, for every $1 \le i < N$. As $U(i,j) \ge U(i,j+1)$ for every $i < j < N$, the following lemma holds.

▶ **Lemma 8.** *$U(i,j) \le \lambda$ iff $\mu_i \le j$ (see also Figure 3).*

Moreover, as $U(i,j) \le U(i+1,j)$, we have that $\mu_i \le \mu_{i+1}$. Therefore, our algorithm can compute all the indices μ_i in $O(N)$ time by scanning all the vertices of P from 1 to N.

We introduce some new notation useful to describe our algorithm. We define r_i as the maximum index such that $i < r_i \le N$ and $\omega(i) + d(i,r_i) + \omega(r_i) \le \lambda$. If such an index does not exist, we set $r_i = i$. Similarly, we define ℓ_N as the minimum index such that $1 \le \ell_N < N$ and $\omega(\ell_N) + d(\ell_N,N) \le \lambda$. If such an index does not exist, we set $\ell_N = N$. Observe that if $j \le r_i$, then, using the node-triangle inequality, $\omega(i) + d(i,j) + \omega(j) \le \omega(i) + d(i,j) + d(j,r_i) + \omega(r_i) = \omega(i) + d(i,r_i) + \omega(r_i) \le \lambda$. Therefore,

$$\omega(i) + d(i,j) + \omega(j) \le \lambda \text{ iff } j \le r_i. \tag{1}$$

Similarly, if $\ell_N \le i$, then, using the node-triangle inequality, $\omega(i) + d(i,N) \le \omega(\ell_N) + d(\ell_N,i) + d(i,N) = \omega(\ell_N) + d(\ell_N,N) \le \lambda$. Therefore,

$$\omega(i) + d(i,N) \le \lambda \text{ iff } \ell_N \le i. \tag{2}$$

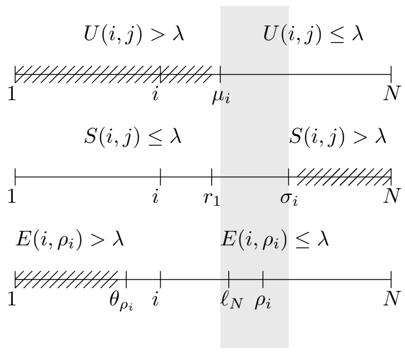

Figure 3 An example showing the properties satisfied by the functions $U(i,j)$, $S(i,j)$, and $E(i,\rho_i)$. The example shows a case in which ρ_i is defined. We observe that the search version of WDoap admits a feasible solution consisting of a shortcut adjacent to i iff there exists an index j belonging to the shaded area such that $C(i,j) \le \lambda$. Furthermore, among all the possible choices, ρ_i is the one that minimizes the value $C(i,j)$.

The algorithm computes all the indices r_i, with $1 \le i < N$, and the index ℓ_N. Since $\omega(i) \ge \omega(i+1) - d(i,i+1)$, we have that $r_i \le r_{i+1}$. Therefore, all the r_i's can be computed in $O(N)$ time by scanning all the vertices of P in order from 1 to N. Clearly, also ℓ_N can be computed in $O(N)$ time by scanning all the vertices of P in order from N downto 1. As $d(1,N) > \lambda$, we have that $r_1 < N$ and $\ell_N > 1$. We define the following two functions

$$\bar{S}(i,j) := d(1,i) + c(i,j) + d(r_1+1,j) + \omega(r_1+1)$$

and

$$\bar{E}(i,j) := d(j,N) + c(i,j) + d(i,\ell_N - 1) + \omega(\ell_N - 1).$$

Observe that both $\bar{S}(i,j)$ and $\bar{E}(i,j)$ can be computed in constant time. Moreover, using the graph-triangle inequality, we have that

- if $r_1 < j$, then $\bar{S}(i,j) \le \bar{S}(i,j+1)$;
- if $i < \ell_N$, then $\bar{E}(i,j) \le \bar{E}(i+1,j)$.

As the following lemma shows, the values $\bar{S}(i,j)$ and $\bar{E}(i,j)$ can be used to understand whether $S(i,j) \le \lambda$ and $E(i,j) \le \lambda$, respectively.

▶ **Lemma 9.** *If $U(i,j) \le \lambda$, then:*

- *$S(i,j) \le \lambda$ iff $i \le r_1$ and $\bar{S}(i,j) \le \lambda$;*
- *$E(i,j) \le \lambda$ iff $\ell_N \le j$ and $\bar{E}(i,j) \le \lambda$.*

Let $i < \sigma_i \le N$ be the maximum index, or i if such an index does not exist, such that $\bar{S}(i,\sigma_i) \le \lambda$. Analogously, let $1 \le \theta_j < j$ be the minimum index, or j if such an index does not exist, such that $\bar{E}(\theta_j,j) \le \lambda$. Our algorithm computes all the indices σ_i, with $1 \le i < N$, and the indices θ_j, with $1 < j \le N$. By the graph-triangle inequality, $\bar{S}(i,j) \le \bar{S}(i+1,j)$ as well as $\bar{E}(i,j) \le \bar{E}(i,j-1)$. As a consequence, $\sigma_{i+1} \le \sigma_i$ and $\theta_{j-1} \ge \theta_j$. Therefore, all the σ_i's can be computed in $O(N)$ time by scanning all the vertices of P in order from 1 to N. Similarly, all the θ_j's can be computed in $O(N)$ time by scanning all the vertices of P in order from N downto 1. The following lemma holds.

▶ **Lemma 10.** *If $U(i,j) \le \lambda$, then:*

- *$S(i,j) \le \lambda$ iff $i \le r_1$ and $j \le \sigma_i$ (see also Figure 3);*
- *$E(i,j) \le \lambda$ iff $\ell_N \le j$ and $\theta_j \le i$ (see also Figure 3).*

Let ρ_i be the minimum index, or $\perp$ if such an index does not exist, such that $\mu_i \leq \rho_i \leq \sigma_i$ and $i \geq \theta_{\rho_i}$. The algorithm computes ρ_i, for every $i = 1, \ldots, N$. Since $\mu_i \leq \mu_{i+1}$, $\sigma_{i+1} \leq \sigma_i$, and $\theta_{j-1} \geq \theta_j$, all the indices ρ_i can be computed in $O(N)$ time. The following lemma holds.

▶ **Lemma 11.** *Let $\langle P, \delta, \omega, c\rangle$ be an instance of* WDoap*, where c is a graph-metric and ω satisfies the node-triangle inequality, and let $\lambda > 0$. There exists an index $1 \leq i < N$, such that ρ_i is defined and $C(i, \rho_i) \leq \lambda$ iff the search version of* WDoap *on input instance $\langle P, \delta, \omega, c, \lambda\rangle$ admits a feasible solution.*

In the following we show how to check whether $C(i, \rho_i) \leq \lambda$ in constant time after an $O(N)$ time preprocessing. For every $1 \leq i < N$ such that $r_i < N$, the algorithm computes

$$\Delta(i) = \lambda - \omega(i) + d(i, r_i + 1) - \omega(r_i + 1).$$

Moreover, the algorithm computes $\Delta_{\min} = \min_{1 \leq i < N} \Delta(i)$. Finally, for every $i = 1, \ldots, N$ for which ρ_i is defined, our algorithm checks whether $d(i, \rho_i) + c(i, \rho_i) \leq \Delta_{\min}$. If there exists i such that $d(i, \rho_i) + c(i, \rho_i) \leq \Delta_{\min}$, then our algorithm returns (i, ρ_i). If this is not the case, then our algorithm returns $\perp$. The following lemma proves the correctness of our algorithm.

▶ **Lemma 12.** *Let $\langle P, \delta, \omega, c\rangle$ be an instance of* WDoap*, where c is a graph-metric and ω satisfies the node-triangle inequality, and let $\lambda > 0$. The search version of* WDoap *on input instance $\langle P, \delta, \omega, c, \lambda\rangle$ admits a feasible solution iff there exists an index $1 \leq i < N$, such that ρ_i is defined and $d(i, \rho_i) + c(i, \rho_i) \leq \Delta_{\min}$.*

We can conclude this section with the following theorem.

▶ **Theorem 13.** *Let $\langle P, \delta, \omega, c\rangle$ be an instance of* WDoap*, where c is a graph-metric and ω satisfies the node-triangle inequality, and let $\lambda > 0$. The search version of* WDoap *on input instance $\langle P, \delta, \omega, c, \lambda\rangle$ can be solved in $O(N)$ time and space.*

5 The algorithm for WDoap

In this section we design an efficient $O(N \log N)$ time and $O(N)$ space algorithm that finds an optimal solution for instances $\langle P, \delta, \omega, c\rangle$ of WDoap, where c is a graph-metric and ω satisfies the node-triangle inequality. In the rest of the paper we denote by D^* the diameter of an optimal solution to the problem instance and, of course, we assume that D^* is not known by the algorithm. For the rest of this section, unless stated otherwise, i and j are two fixed indices such that $1 \leq i < j \leq N$. Similarly to the notation already used in the previous section, we define r_i as the maximum index such that $i < r_i \leq N$ and $\omega(i) + d(i, r_i) + \omega(r_i) \leq D^*$. If such an index does not exist, then $r_i = i$. Analogously, we define ℓ_N as the minimum index such that $1 \leq \ell_N < N$ and $\omega(\ell_N) + d(\ell_N, N) \leq D^*$. If such an index does not exist, then $\ell_N = N$. Our algorithm consists of the following three phases:

1. a precomputation phase in which the algorithm computes all the indices r_i, with $1 \leq i < N$, and the index ℓ_N;
2. a search-space reduction phase in which the algorithm reduces the size of the solution search space from $\binom{N}{2}$ to $N - 1$ candidates;
3. an optimal-solution selection phase in which the algorithm builds a data structure that is used to evaluate the leftover $N - 1$ solutions in $O(\log N)$ time per solution.

Each of the three phases requires $O(N \log N)$ time and $O(N)$ space; furthermore, they all make use of the linear time algorithm for the search version of WDoap black-box. In the following we assume that $d(1, N) > D^*$, as otherwise, any shortcut returned by our algorithm would be an optimal solution.

5.1 The precomputation phase

We perform a binary search over the indices from 1 to N and use the linear time algorithm for the search version of WDOAP to compute the maximum index ℓ_N in $O(N \log N)$ time and $O(N)$ space. Indeed, when our binary search considers the index k as a possible choice of ℓ_N, it is enough to call the linear time algorithm for the search version of WDOAP with parameter $\lambda = \omega(k) + d(k, N)$ and see whether the algorithm returns either a feasible solution or $\perp$. Due to the node-triangle inequality, in the former case we know that $\ell_N \leq k$, while in the latter case we know that $\ell_N > k$.

Now we describe how to compute all the indices r_i. Because of the node-triangle inequality $r_i < N$ iff $i < \ell_N$. Therefore, we only have to describe how to compute r_i for every $i < \ell_N$, since if $i \geq \ell_N$, then $r_i = N$. We use the linear time algorithm for the search version of WDOAP and perform a binary search over the set of sorted values $\{\omega(i) + d(i, i+1) + \omega(i+1) \mid 1 \leq i < \ell_N\}$ to compute the largest value of the set that is less than or equal to D^*, if any. This allows us to compute, in $O(N \log N)$ time and $O(N)$ space, the set of all indices $i < \ell_N$ for which $r_i = i$. Now, for every index $i < \ell_N$ for which $r_i > i$, we set $a_i = i + 1$ and $b_i = N$. Observe that $a_i \leq r_i \leq b_i$. Next, using a two-round binary search, we restrict all the intervals $[a_i, b_i]$'s by updating both a_i and b_i while maintaining the invariant property $a_i \leq r_i \leq b_i$ at the same time.

Let X be the set of indices i, with $1 \leq i < \ell_N$, for which $b_i \geq a_i + 2$. The first round of the binary search ends exactly when X becomes empty. Each iteration of the first round works as follows. For every $i \in X$, the algorithm computes the median index $m_i = \lfloor (a_i + b_i)/2 \rfloor$. Next the algorithm computes the *weighted* median of the m_i's, say m_τ, where the weight of m_i is equal to $b_i - a_i$. Let

$$X_\tau^+ = \{i \in X \mid \omega(i) + d(i, m_i) + \omega(m_i) \geq \omega(\tau) + d(\tau, m_\tau) + \omega(m_\tau)\}$$

and

$$X_\tau^- = \{i \in X \mid \omega(i) + d(i, m_i) + \omega(m_i) \leq \omega(\tau) + d(\tau, m_\tau) + \omega(m_\tau)\}.$$

Observe that $X = X_\tau^+ \cup X_\tau^-$ and $\tau \in X^+, X^-$.

Now we call the linear time algorithm for the search version of WDOAP with parameter $\lambda = \omega(\tau) + d(\tau, m_\tau) + \omega(m_\tau)$. If the algorithm finds two indices such that $D(i, j) \leq \lambda$, then we know that $D^* \leq \lambda$ and therefore, for every $i \in X_\tau^+$, we update b_i by setting it equal to m_i. If the algorithm outputs $\perp$, then we know that $D^* > \lambda$ and therefore, for every $i \in X_\tau^-$, we update a_i by setting it equal to m_i. We observe that in either case, the invariant property $a_i \leq r_i \leq b_i$ is kept because of (1). An iteration of the first round of the binary search ends right after the removal of all the indices i such that $b_i = a_i + 1$ from X. Notice that, in the worst case, the overall sum of the intervals widths at the end of a single iteration is (almost) $3/4$ times the same value computed at the beginning of the iteration. This implies that the first round of the binary search ends after $O(\log N)$ iterations. Furthermore, the time complexity of each iteration is $O(N)$. Therefore, the overall time needed for the first round of the binary search is $O(N \log N)$.

The second round of the binary search works as follows. Because $a_i \leq r_i \leq b_i$ and $b_i \leq a_i + 1$ for every $i < \ell_N$ such that $i < r_i$, we have that r_i is equal to either a_i or b_i. To understand whether either $r_i = a_i$ or $r_i = b_i$, we sort the (at most) $2N$ values

$$\Upsilon = \bigcup_{i < \ell_N,\, i < r_i} \{\omega(i) + d(i, a_i) + \omega(a_i), \omega(i) + d(i, b_i) + \omega(b_i)\}$$

and use binary search, together with the linear time algorithm for the search version of WDoap, to compute the two consecutive distinct values $D^+, D^- \in \Upsilon$ such that $D^- < D^* \leq D^+$ (if D^- does not exist, then we assume it to be equal to 0). Finally, we use the two values D^+ and D^- to select either a_i or b_i. More precisely, if $a_i = b_i$, then $r_i = a_i$. If $a_i \neq b_i$, then by the choice of D^- and D^+, either $D^- < \omega(i) + d(i, a_i) + \omega(a_i) \leq D^+$ (i.e., $r_i = a_i$) or $D^- < \omega(i) + d(i, b_i) + \omega(b_i) \leq D^+$ (i.e., $r_i = b_i$). The time and space complexities of the second round of the binary search are $O(N \log N)$ and $O(N)$, respectively. We have proved the following lemma.

▶ **Lemma 14.** *The precomputation phase takes $O(N \log N)$ time and $O(N)$ space.*

5.2 The search-space reduction phase

In the search-space reduction phase the algorithm computes a set of $N - 1$ candidates as optimal shortcut in $O(N \log N)$ time and $O(N)$ space. Let $f(i,j) := \max\big\{U(i,j), \bar{E}(i,j)\big\}$. Since both $U(i,j)$ and $\bar{E}(i,j)$ are monotonically non-increasing w.r.t. j,[3] $f(i,j)$ is monotonically non-increasing w.r.t. j. For every $1 \leq i < N$, our algorithm computes the minimum index $1 < \psi_i \leq N$, if any, such that $f\big(i, \psi_i\big) \leq D^*$. As both $S(i,j)$ and $C(i,j)$ are monotonically non-decreasing w.r.t. j, it follows that the set $\big\{\big(i, \psi_i\big) \mid 1 \leq i < N\big\}$ contains an optimal solution to the problem instance.

We compute all the indices ψ_i's using a two-round binary search techique similar to the one we already used in the precomputation phase. First, we set $a_i = i + 1$ and $b_i = N$, for every $1 \leq i < N$. Observe that $a_i \leq \psi_i \leq b_i$. In the two-round binary search, we restrict all the intervals $[a_i, b_i]$'s by updating both a_i and b_i while maintaining the invariant property $a_i \leq \psi_i \leq b_i$ at the same time.

Let X be the set of indices i for which $b_i \geq a_i + 2$. The first round of the binary search ends exactly when X becomes empty. Each iteration of the first round works as follows. For every $i \in X$, the algorithm computes the median index $m_i = \big\lfloor (a_i + b_i)/2 \big\rfloor$. Next the algorithm computes the *weighted* median of the m_i's, say m_τ, where the weight of m_i is equal to $b_i - a_i$. Let

$$X_\tau^+ = \big\{i \in X \mid f(i, m_i) \geq f(\tau, m_\tau)\big\} \qquad \text{and} \qquad X_\tau^- = \big\{i \in X \mid f(i, m_i) \leq f(\tau, m_\tau)\big\}.$$

Observe that $X = X_\tau^+ \cup X_\tau^-$; moreover, $\tau \in X^+, X^-$.

Now we call the linear time algorithm for the search version of WDoap with parameter $\lambda = f(\tau, m_\tau)$. If the algorithm finds two indices such that $D(i,j) \leq \lambda$, then we know that $D^* \leq f(\tau, m_\tau)$ and therefore, since $f(i,j) \leq f(i, j+1)$, for every $i \in X_\tau^+$, we update b_i by setting it equal to m_i. If the algorithm outputs $\perp$, then we know that $D^* > f(\tau, m_\tau)$ and therefore, by monotonicity of f, for every $i \in X_\tau^-$, we update a_i by setting it equal to m_i. We observe that in either case, the invariant property $a_i \leq \psi_i \leq b_i$ is maintained. An iteration of the first round of the binary search ends right after the removal of all the indices i such that $b_i = a_i + 1$ from X. Notice that, in the worst case, the overall sum of the intervals widths at the end of a single iteration is (almost) 3/4 times the same value computed at the beginning of the iteration. This implies that the first round of the binary search ends after $O(\log N)$ iterations. Furthermore, both the time and space complexities of each iteration is $O(N)$. Therefore, the overall time needed for the first round of the binary search is $O(N \log N)$.

[3] Observe that $E(i,j) = \max\{\bar{E}(i,j), \omega(\ell_N) + d(\ell_N, N)\}$ because of the node-triangle inequality. However, since we know that $\omega(\ell_N) + d(\ell_N, N) \leq D^*$ by definition, we can check whether $E(i,j) \leq D^*$ by simply evaluating $\bar{E}(i,j)$.

The second round of the binary search works as follows. Because $a_i \leq \psi_i \leq b_i$ and $b_i \leq a_i + 1$, ψ_i is either equal to a_i or to b_i. To understand whether either $\psi_i = a_i$ or $\psi_i = b_i$, we sort the (at most) $2N$ values $\Upsilon = \bigcup_{1 \leq i < N} \{f(i, a_i), f(i, b_i)\}$ and use binary search, together with the linear time algorithm for the search version of WDOAP, to compute the two consecutive distinct values $D^+, D^- \in \Upsilon$ such that $D^- < D^* \leq D^+$ (if D^- does not exist, then we assume it to be equal to 0). Finally, we use the two values D^+ and D^- to select either a_i or b_i. More precisely, if $a_i = b_i$, then $\psi_i = a_i$. If $a_i \neq b_i$, then by the choice of D^- and D^+, either $D^- < f(i, a_i) \leq D^+$ (i.e., $\psi_i = a_i$) or $D^- < f(i, b_i) \leq D^+$ (i.e., $\psi_i = b_i$). The time and space complexities of the second round of the binary search are $O(N \log N)$ and $O(N)$, respectively. We have proved the following lemma.

▶ **Lemma 15.** *The search-space reduction phase takes $O(N \log N)$ time and $O(N)$ space. Furthermore, there exists a shortcut (i^*, ψ_{i^*}) such that $D\big(i^*, \psi_{i^*}\big) = D^*$.*

5.3 The optimal-solution selection phase

In the optimal-solution selection phase, we build a data structure in $O(N \log N)$ time and use it to evaluate the quality of the $N - 1$ candidates $(1, \psi_1), \ldots, (N-1, \psi_{N-1})$ in $O(\log N)$ time per candidate. For every $k = 1, \ldots, N$, we define

$$\phi_k(x) := \omega(k) + \max\big\{d(k, r_k) + \omega(r_k), x - d(k, r_k + 1) + \omega(r_k + 1)\big\}.$$

Let $\mathcal{U}(x) := \max\big\{\phi_k(x) \mid 1 \leq k < \ell_N\big\}$ be the *upper envelope* of all the functions $\phi_k(x)$. Observe that each $\phi_k(x)$ is itself the upper envelope of two linear functions. Therefore, $\mathcal{U}(x)$ is the upper envelope of at most $2N - 2$ linear functions. In [12] it is shown how to compute the upper envelope of a set of $O(N)$ linear functions in $O(N \log N)$ time and $O(N)$ space. In the same paper it is also shown how the value $\mathcal{U}(x)$ can be computed in $O(\log N)$ time, for any $x \in \mathbb{R}$.

We denote by $x_i = d(i, \psi_i) + c(i, \psi_i)$ the overall weight of the edges of the unique cycle in $P + (i, \psi_i)$. For every $1 \leq i < N$, we compute the value

$$\eta_i = \max\Big\{U\big(i, \psi_i\big), S\big(i, \psi_i\big), E\big(i, \psi_i\big), \mathcal{U}(x_i)\Big\}. \tag{3}$$

The algorithm computes the index α that minimizes η_α and returns the shortcut (α, ψ_α) .

▶ **Lemma 16.** *For every i, with $1 \leq i < N$, $C(i, \psi_i) \leq \mathcal{U}(x_i)$.*

Let i^* be the index such that $D(i^*, \psi_{i^*}) = D^*$, whose existence is guaranteed by Lemma 15. The following lemma holds.

▶ **Lemma 17.** *$\mathcal{U}(x_{i^*}) \leq D^*$.*

We can finally conclude this section by stating the main results of this paper.

▶ **Theorem 18.** WDOAP *can be solved in $O(N \log N)$ time and $O(N)$ space.*

▶ **Theorem 19.** DOAP *on trees embedded in a (graph-)metric space can be solved in $O(n \log n)$ time and $O(n)$ space.*

References

1 Noga Alon, András Gyárfás, and Miklós Ruszinkó. Decreasing the diameter of bounded degree graphs. *Journal of Graph Theory*, 35(3):161–172, 2000. doi:10.1002/1097-0118(200011)35:3\%3C161::AID-JGT1\%3E3.0.CO;2-Y.

2 Davide Bilò, Luciano Gualà, and Guido Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theor. Comput. Sci.*, 417:12–22, 2012. doi:10.1016/j.tcs.2011.05.014.

3 Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, and Michiel H. M. Smid. Minimizing the Continuous Diameter when Augmenting Paths and Cycles with Shortcuts. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016*, volume 53 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.SWAT.2016.27.

4 Jean-Lou De Carufel, Carsten Grimm, Stefan Schirra, and Michiel H. M. Smid. Minimizing the Continuous Diameter When Augmenting a Tree with a Shortcut. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017*, volume 10389 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2017. doi:10.1007/978-3-319-62127-2_26.

5 Victor Chepoi and Yann Vaxès. Augmenting Trees to Meet Biconnectivity and Diameter Constraints. *Algorithmica*, 33(2):243–262, 2002. doi:10.1007/s00453-001-0113-8.

6 F. R. K. Chung and M. R. Garey. Diameter bounds for altered graphs. *Journal of Graph Theory*, 8(4):511–534, 1984. doi:10.1002/jgt.3190080408.

7 Erik D. Demaine and Morteza Zadimoghaddam. Minimizing the Diameter of a Network Using Shortcut Edges. In Haim Kaplan, editor, *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory*, volume 6139 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2010. doi:10.1007/978-3-642-13731-0_39.

8 Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. Augmenting Graphs to Minimize the Diameter. *Algorithmica*, 72(4):995–1010, 2015. doi:10.1007/s00453-014-9886-4.

9 Yong Gao, Donovan R. Hare, and James Nastos. The parametric complexity of graph diameter augmentation. *Discrete Applied Mathematics*, 161(10-11):1626–1631, 2013. doi:10.1016/j.dam.2013.01.016.

10 Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel H. M. Smid, and Fabian Stehn. Fast Algorithms for Diameter-Optimally Augmenting Paths. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015*, volume 9134 of *Lecture Notes in Computer Science*, pages 678–688. Springer, 2015. doi:10.1007/978-3-662-47672-7_55.

11 Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel H. M. Smid, and Fabian Stehn. Fast Algorithms for Diameter-Optimally Augmenting Paths and Trees. *CoRR*, abs/1607.05547, 2016. arXiv:1607.05547.

12 John Hershberger. Finding the Upper Envelope of n Line Segments in $O(n \log n)$ Time. *Inf. Process. Lett.*, 33(4):169–174, 1989. doi:10.1016/0020-0190(89)90136-1.

13 Toshimasa Ishii. Augmenting Outerplanar Graphs to Meet Diameter Requirements. *Journal of Graph Theory*, 74(4):392–416, 2013. doi:10.1002/jgt.21719.

14 Chung-Lun Li, S.Thomas McCormick, and David Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum diameter edge addition problems. *Operations Research Letters*, 11(5):303–308, 1992.

15 Nimrod Megiddo. Applying Parallel Computation Algorithms in the Design of Serial Algorithms. *J. ACM*, 30(4):852–865, 1983. doi:10.1145/2157.322410.

16 Eunjin Oh and Hee-Kap Ahn. A Near-Optimal Algorithm for Finding an Optimal Shortcut of a Tree. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia*, volume 64 of *LIPIcs*, pages 59:1–59:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.ISAAC.2016.59`.

17 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. `doi:10.1002/jgt.3190110315`.

18 Haitao Wang. An Improved Algorithm for Diameter-Optimally Augmenting Paths in a Metric Space. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 15th International Symposium, WADS 2017*, volume 10389 of *Lecture Notes in Computer Science*, pages 545–556. Springer, 2017. `doi:10.1007/978-3-319-62127-2_46`.