Algorithms for Diameters of Unicycle Graphs and Diameter-Optimally Augmenting Trees

Haitao Wang
Department of Computer Science, Utah State University, Logan, UT 84322, USA.
haitao.wang@usu.edu

Yiming Zhao
Department of Computer Science, Utah State University, Logan, UT 84322, USA.
yiming.zhao@usu.edu

Abstract

We consider the problem of computing the diameter of a unicycle graph (i.e., a graph with a unique cycle). We present an $O(n)$ time algorithm for the problem, where $n$ is the number of vertices of the graph. This improves the previous best $O(n \log n)$ time solution [Oh and Ahn, ISAAC 2016]. Using this algorithm as a subroutine, we solve the problem of adding a shortcut to a tree so that the diameter of the new graph (which is a unicycle graph) is minimized; our algorithm takes $O(n^2 \log n)$ time and $O(n)$ space. The previous best algorithms solve the problem in $O(n^2 \log^3 n)$ time and $O(n)$ space [Oh and Ahn, ISAAC 2016], or in $O(n^2)$ time and $O(n^2)$ space [Bilò, ISAAC 2018].

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases diameter, unicycle graphs, augmenting trees, shortcuts

Funding This research was supported in part by NSF under Grant CCF-2005323.

1 Introduction

Let $G$ be a graph of $n$ vertices such that each edge has a positive length. A shortest path connecting two vertices $s$ and $t$ in $G$ is a path of minimum total edge length; the length of the shortest path is also called the distance between $s$ and $t$ in $G$. The diameter of $G$ is the maximum distance between all pairs of vertices of $G$. $G$ is a unicycle graph if it has only one cycle, i.e., $G$ is a tree plus an additional edge.

We consider the problem of computing the diameter of a unicycle graph $G$. Previously, Oh and Ahn [20] solved the problem in $O(n \log n)$ time, where $n$ is the number of vertices of $G$. We present an improved algorithm of $O(n)$ time. Using our new algorithm, we also solve the diameter-optimally augmenting tree (DOAT for short) problem, defined as follows.

Let $T$ be a tree of $n$ vertices such that each edge has a positive length. We want to add a new edge (called shortcut) to $T$ such that the new graph (which is a unicycle graph) has the minimum diameter. We assume that there is an oracle that returns the length of any given shortcut in $O(1)$ time. Previously, Oh and Ahn [20] solved the problem in $O(n^2 \log^3 n)$ time and $O(n)$ space, and Bilò [3] reduced the time to $O(n^2)$ but the space increases to $O(n^2)$. As observed by Oh and Ahn [20], the problem has an $\Omega(n^2)$ lower bound on the running time as all $\Theta(n^2)$ possible shortcuts have to be checked in order to find an optimal shortcut. Hence, Bilò’s algorithm is time-optimal. In this paper, we propose an algorithm with a better time and space trade-off, and our algorithm uses $O(n^2 \log n)$ time and $O(n)$ space.

* A preliminary version of this paper appears in the Proceedings of the 15th International Conference and Workshops on Algorithms and Computation (WALCOM 2021).

1 Corresponding author.
1.1 Related work

The diameter is an important measure of graphs and computing it is one of the most fundamental algorithmic graph problems. For general graphs or even planar graphs, the only known way to compute the diameter is to first solve the all-pair-shortest-path problem (i.e., compute the distances of all pairs of vertices of the graph), which inherently takes \(\Omega(n^2)\) time, e.g., \([11,25]\). Better algorithms exist for special graphs. For example, the diameter of a tree can be computed in linear time, e.g., by first computing its center \([19]\). If \(G\) is an outerplanar graph and all edges have the same length, its diameter can be computed in linear time \([10]\). The diameter of interval graphs (with equal edge lengths) can also be computed in linear time \([21]\). Our result adds the unicycle graph (with different edge lengths) to the linear-time solvable graph category.

The DOAT problem and many of its variations enjoy an increasing interest in the research community. If the tree \(T\) is embedded in a metric space (so that the triangle inequality holds for edge lengths), Große et al. \([15]\) first solved the problem in \(O(n^2 \log n)\) time. Bilò \([3]\) later gave an \(O(n \log n)\) time and \(O(n)\) space algorithm, and another \((1 + \epsilon)\)-approximation algorithm of \(O(n + \frac{1}{\epsilon} \log \frac{1}{\epsilon})\) time and \(O(n + \frac{1}{\epsilon})\) space for any \(\epsilon > 0\). A special case where \(T\) is a path embedded in a metric space was first studied by Große et al. \([14]\), who gave an \(O(n \log^3 n)\) time algorithm, and the algorithm was later improved to \(O(n \log n)\) time by Wang \([23]\). Hence, Bilò’s work \([3]\) generalizes Wang’s result \([23]\) to trees.

A variant of the DOAT problem which aims to minimize the continuous diameter, i.e., the diameter of \(T\) is measured with respect to all the points of the tree (including the points in the interior of the edges), has also been studied. If \(T\) is a path embedded in the Euclidean plane, De Carufel et al. \([5]\) solved the problem in \(O(n)\) time. If \(T\) is a tree embedded in a metric space, De Carufel et al. \([6]\) gave an \(O(n \log n)\) time algorithm. If \(T\) is a general tree, Oh and Ahn \([20]\) solved the problem in \(O(n^2 \log^3 n)\) time and \(O(n)\) space.

The DOAT problem is to minimize the diameter. The problem of minimizing the radius was also considered. For the case where \(T\) is a path embedded in a metric space, Johnson and Wang \([17]\) presented a linear time algorithm which adds a shortcut to \(T\) so that the radius of the resulting graph is minimized. The radius considered in \([17]\) is defined with respect to all points of \(T\), not just the vertices. Wang and Zhao \([24]\) studied the same problem with radius defined with respect to only the vertices, and they gave a linear time algorithm.

The more general problem in which one wants to add \(k\) shortcuts to a graph to minimize the diameter is NP-hard \([22]\) and some variations are even \(W[2]\)-hard \([12,13]\). Approximation algorithms have been proposed \([4,7,9,12,18]\). The upper and lower bounds on the values of diameters of certain augmented graphs were also studied, e.g., \([1,8,16]\). Bae et al. \([2]\) considered the problem of adding \(k\) shortcuts to a circle in the plane to minimize the diameter of the resulting graph.

1.2 Our approach

To compute the diameter of a unicycle graph \(G\), Oh and Ahn \([20]\) reduces the problem to a geometric problem and then uses a one-dimensional range tree to solve the problem. We take a completely different approach. Let \(C\) be the unique cycle of \(G\). We define certain “domination” relations on the vertices of \(C\) so that if a vertex \(v\) is dominated by another vertex then \(v\) is not important to the diameter. We then present a pruning algorithm to find all undominated vertices (and thus those dominated vertices are “pruned”); it turns out that finding the diameter among the undominated vertices is fairly easy. In this way, we compute the diameter of \(G\) in linear time.
For the DOAT problem on a tree \( T \), Oh and Ahn \[20\] considered all possible shortcuts of \( T \) by following an Euler tour of \( T \); they used the aforementioned one-dimensional range tree to update the diameter for the next shortcut. Bilò’s method \[3\] is to transform the problem to adding a shortcut to a path whose edge lengths satisfy a property similar in spirit to the triangle inequality (called graph-triangle inequality) and then the problem on \( P \) can be solved by applying the \( O(n \log n) \) time algorithm for trees in metric space \[3\]. Unfortunately, the problem transformation algorithm relies on using \( O(n^2) \) space to store the lengths of all possible \( \Theta(n^2) \) shortcuts of \( T \). The algorithm has to consider all these \( \Theta(n^2) \) shortcut lengths in a global manner and thus it inherently uses \( \Omega(n^2) \) space. Note that Bilò’s method \[3\] does not need an algorithm for computing the diameter of a unicycle graph.

We propose a novel approach. We first compute a diametral path \( P \) of \( T \). Then we reduce the DOAT problem on \( T \) to finding a shortcut for \( P \). To this end, we consider vertices of \( P \) individually. For each vertex \( v \) of \( P \), we want to find an optimal shortcut with the restriction that it must connect \( v \), dubbed a \( v \)-shortcut. For this, we define a “domination” relation on all \( v \)-shortcuts and we show that these shortcuts dominated by others are not important. We then design a pruning algorithm to find all shortcuts that are not dominated by others; most importantly, these undominated shortcuts have certain monotonicity properties that allow us to perform binary search to find an optimal \( v \)-shortcut by using our diameter algorithm for unicycle graphs as a subroutine. With these effort, we find an optimal \( v \)-shortcut in \( O(n \log n) \) time and \( O(n) \) space. The space can be reused for computing optimal \( v \)-shortcuts of other vertices of \( P \). In this way, the total time of the algorithm is \( O(n^2 \log n) \) and the space is \( O(n) \).

**Outline.** In the following, we present our algorithm for computing the diameter of a unicycle graph in Section \[2\]. Section \[3\] is concerned with the DOAT problem.

## 2 Computing the Diameter of Unicycle Graphs

In this section, we present our linear time algorithm for computing the diameter of unicycle graphs.

For a subgraph \( G' \) of a graph \( G \) and two vertices \( u \) and \( v \) from \( G' \), we use \( \pi_{G'}(u,v) \) to denote a shortest path from \( u \) to \( v \) in \( G' \) and use \( d_{G'}(u,v) \) to denote the length of the path. We use \( \Delta(G) \) to denote the diameter of \( G \). A pair of vertices \( (u,v) \) is called a diametral pair and \( \pi_{G}(u,v) \) is called a diametral path if \( d_{G}(u,v) = \Delta(G) \).

In the following, let \( G \) be a unicycle graph of \( n \) vertices. Our goal is to compute the diameter \( \Delta(G) \) (along with a diametral pair). Let \( C \) denote the unique cycle of \( G \).

### 2.1 Observations

Removing all edges of \( C \) (while keeping its vertices) from \( G \) results in several connected components of \( G \). Each component is a tree that contains a vertex \( v \) of \( C \); we use \( T(v) \) to denote the tree. Let \( v_1, v_2, \ldots, v_m \) be the vertices ordered clockwise on \( C \). Let \( T(G) = \{T(v_i) \mid 1 \leq i \leq m\} \). Note that the sets of vertices of all trees of \( T(G) \) form a partition of the vertex set of \( G \).

Consider a diametral pair \( (u^*, v^*) \) of \( G \). There are two cases: (1) both \( u^* \) and \( v^* \) are in the same tree of \( T(G) \); (2) \( u^* \) and \( v^* \) are in two different trees of \( T(G) \). To handle the first case, we compute the diameter of each tree of \( T(G) \), which can be done in linear time. Computing the diameters for all trees takes \( O(n) \) time. The longest diameter of these trees is the diameter of \( G \). In the following, we focus on the second case.
Suppose $T(v_i)$ contains $u^*$ and $T(v_j)$ contains $v^*$ for $i \neq j$. Observe that the diametral path $\pi_G(u^*, v^*)$ is the concatenation of the following three paths: $\pi_{T(v_i)}(u^*, v_i), \pi_C(v_i, v_j)$, and $\pi_{T(v_j)}(v_j, v^*)$. Further, $u^*$ is the farthest vertex in $T(v_i)$ from $v_i$; the same holds for $v^*$ and $T(v_j)$. On the basis of these observations, we introduce some concepts as follows.

For each vertex $v_i \in C$, we define a weight $w(v_i)$ as the length of the path from $v_i$ to its farthest vertex in $T(v_i)$. The weights for all vertices on $C$ can be computed in total $O(n)$ time. With this definition in hand, we have $\Delta(G) = \max_{1 \leq i < j \leq m}(w(v_i) + d_C(v_i, v_j) + w(v_j))$. We say that $(v_i, v_j)$ is a vertex-weighted diametral pair of $C$ if $T(v_i)$ contains $u^*$ and $T(v_j)$ contains $v^*$ for a diametral pair $(u^*, v^*)$ of $G$. To compute $\Delta(G)$, it suffices to find a vertex-weighted diameter pair of $C$.

We introduce a domination relation for vertices on $C$.

**Definition 1.** For two vertices $v_i, v_j \in C$, we say that $v_i$ dominates $v_j$ if $w(v_i) > w(v_j) + d_C(v_i, v_j)$.

The following lemma shows that if a vertex is dominated by another vertex, then it is not “important”.

**Lemma 2.** For two vertices $v_i$ and $v_j$ of $C$, if $v_i$ dominates $v_j$, then $v_j$ cannot be in any vertex-weighted diametral pair of $C$ unless $(v_i, v_j)$ is such a pair.

**Proof.** We assume that $(v_i, v_j)$ is not a vertex-weighted diametral pair. Assume to the contrary that $(v_k, v_j)$ is a vertex-weighted diametral pair of $C$. Then, $k \neq i$ and $\Delta(G) = w(v_k) + d_C(v_k, v_j) + w(v_j)$. Note that $d_C(v_k, v_j) \leq d_C(v_k, v_i) + d_C(v_i, v_j)$ holds. Since $v_i$ dominates $v_j$, we have $w(v_i) > w(v_j) + d_C(v_i, v_j)$ holds. Consequently, we can derive

$$\Delta(G) = w(v_k) + d_C(v_k, v_j) + w(v_j) \leq w(v_k) + d_C(v_k, v_i) + d_C(v_i, v_j) + w(v_j) < w(v_k) + d_C(v_k, v_i) + w(v_i).$$

But this contradicts with the definition of $\Delta(G)$. The lemma thus follows.

### 2.2 A pruning algorithm

In the sequel, we describe a linear time pruning algorithm to find all vertices of $C$ that are dominated by other vertices (and thus those dominated vertices are “pruned”). As will be seen later, the diameter can be easily found after these vertices are pruned.

Let $|C|$ denote the sum of the lengths of all edges of $C$. For any vertex $v_i$ of $C$, define $C_{cw}(v_i)$ as the set of vertices $v_j$ of $C$ such that the path from $v_i$ to $v_j$ counterclockwise along $C$ has length at most $|C|/2$ (e.g., see Fig. 1); define $C_{cw}(v_i)$ as the set of vertices of $C$ not in $C_{cw}(v_i)$. We assume that $v_i$ is in neither subset.

**Lemma 3.** With $O(n)$ time preprocessing, given any two vertices $v_i$ and $v_j$ of $C$, we can do the following in $O(1)$ time: (1) compute $d_C(v_i, v_j)$; (2) determine whether $v_j$ is in $C_{cw}(v_i)$; (3) determine whether $v_i$ and $v_j$ dominate each other.

**Proof.** We first compute the weight $w(v_i)$ for all vertices $v_i \in C$. This can be done in $O(n)$ time. Then, we compute the length $|C|$. Next, by scanning the vertices $v_1, v_2, \ldots, v_m$ on $C$, we compute an array $A[1, \ldots, m]$ with $A[i]$ equal to the length of the path from $v_1$ to $v_i$ clockwise along $C$. Hence, for any $1 \leq i < j \leq m$, $A[j] - A[i]$ is the length of the path from $v_i$ to $v_j$ clockwise along $C$ and $|C| - (A[j] - A[i])$ is the length of the path from $v_i$ to $v_j$ counterclockwise along $C$. Note that $d_C(v_i, v_j) = \min\{A[j] - A[i], |C| - (A[j] - A[i])\}$.
Consider any two vertices $v_i$ and $v_j$ of $C$. Without loss of generality, we assume $i < j$. By comparing $A[j] - A[i]$ with $|C|/2$, we can determine whether $v_j$ is in $C_{ccw}(v_i)$ in $O(1)$ time. As $w(v_i)$ and $w(v_j)$ are both available, whether $v_i$ and $v_j$ dominate each other can be determined in $O(1)$ time.

With Lemma 3 in hand, starting from $v_1$, our pruning algorithm processes the vertices of $C$ from $v_1$ to $v_m$ in order (see Algorithm 1 for the pseudocode). The algorithm maintains a stack $S$, which is empty initially. Consider a vertex $v_i$. If $S = \emptyset$, then we push $v_i$ into $S$. Otherwise, let $v$ be the vertex at the top of $S$. If $v$ is not in $C_{ccw}(v_i)$, then we also push $v_i$ into $S$. Otherwise, we check whether $v$ and $v_i$ dominate each other. If they do not dominate each other, then we push $v_i$ into $S$. Otherwise, if $v_i$ dominates $v$, we pop $v$ out of $S$ and then we continue to pop the new top element $v$ of $S$ out as long as the following three conditions are all satisfied: (1) $S \neq \emptyset$; (2) $v \in C_{ccw}(v_i)$; (3) $v_i$ dominates $v$. Once one of the three conditions is not satisfied, we push $v_i$ into $S$.

After $v_m$ is processed, the first stage of the pruning algorithm is over. In the second stage, we process the vertices in the stack $S$ in a bottom-up manner until a vertex not in $C_{cw}(v_1)$; the processing of a vertex is done in the same way as above (the vertex should be removed from $S$ first). Specifically, let $v_i$ be the vertex at the bottom of $S$. If $v_i$ is not in $C_{cw}(v_1)$, then we stop the algorithm and return the vertices in the current stack $S$. Otherwise, we remove $v_i$ from $S$ and then apply the same processing algorithm as above in the first stage (i.e., begin with checking whether $S$ is empty).

Intuitively, the first stage of the algorithm does a “full-cycle” scan on $C$ while the second stage does a “half-cycle” scan (i.e., the half-cycle clockwise from $v_1$). With Lemma 3, the algorithm can be implemented in $O(n)$ time. The following lemma establishes the correctness of the algorithm.

**Lemma 4.** Let $S$ be the stack after the algorithm is over.
1. Each vertex of $C$ that is not in $S$ is dominated by a vertex in $S$.
2. No two vertices of $S$ dominate each other.

**Proof.** We first proves an observation about two transitive properties of the domination relation, which will be used to prove the lemma.

**Observation.** Let $v_i, v_j, v_k$ be any three vertices of $C$.
1. If $v_i$ dominates $v_j$ and $v_j$ dominates $v_k$, then $v_i$ dominates $v_k$.
Algorithm 1 The pruning algorithm

Function ProcessVertex\( (v_i, S) \):
\[
\text{if } S = \emptyset \text{ then} \\
\quad S.\text{push}(v_i) // \text{Push vertex } v_i \text{ into the stack} \\
\text{end} \\
\text{else} \\
\quad v \leftarrow S.\text{top}() // S.\text{top}() \text{ is the top element of the stack} \\
\quad \text{if } v \notin C_{ccw}(v_i) \text{ then} \\
\quad\quad S.\text{push}(v_i) \\
\quad\text{end} \\
\quad\text{else if } v_i \text{ and } v \text{ do not dominate each other then} \\
\quad\quad S.\text{push}(v_i) \\
\quad\text{end} \\
\quad\text{else if } v_i \text{ dominates } v \text{ then} \\
\quad\quad S.\text{pop}() // \text{Pop the top element out of } S \\
\quad\quad \text{while } S \neq \emptyset \text{ and } S.\text{top}() \in C_{ccw}(v_i) \text{ and } v_i \text{ dominates } S.\text{top}() \text{ do} \\
\quad\quad\quad S.\text{pop}() \\
\quad\quad\text{end} \\
\quad S.\text{push}(v_i) \\
\text{end} \\
\text{end}
\]

Function Main\( (S = \emptyset, C) \):
\[
\text{// the full-cycle scan} \\
\quad S = \emptyset \\
\quad \text{for } i = 1, 2, \ldots, m \text{ do} \\
\quad\quad \text{ProcessVertex}(v_i, S) // \text{Call ProcessVertex function on } v_i \\
\quad \text{end} \\
\text{// the half-cycle scan} \\
\quad v_i \leftarrow \text{the bottom element of } S \\
\quad \text{while } v_i \in C_{cw}(v_1) \text{ do} \\
\quad\quad \text{Remove } v_i \text{ from } S \\
\quad\quad \text{ProcessVertex}(v_i, S) \\
\quad\quad v_i \leftarrow \text{the bottom element of } S \\
\quad\text{end} \\
\quad \text{return } S \\
\text{end}
2. If \( v_i \) and \( v_j \) do not dominate each other, \( v_j \) and \( v_k \) do not dominate each other, and \( d_C(v_i, v_k) = d_C(v_i, v_j) + d_C(v_j, v_k) \), then \( v_i \) and \( v_k \) do not dominate each other.

**Proof of observation.**
1. Since \( v_i \) dominates \( v_j \) and \( v_j \) dominates \( v_k \), we have \( w(v_i) > w(v_j) + d_C(v_i, v_j) \) and \( w(v_j) > w(v_k) + d_C(v_j, v_k) \). Thus, \( w(v_i) > w(v_k) + d_C(v_j, v_k) + d_C(v_i, v_j) \geq w(v_k) + d_C(v_i, v_k) \). Hence, \( v_i \) dominates \( v_k \).
2. As \( v_i \) and \( v_j \) do not dominate each other, \( w(v_i) \leq w(v_j) + d_C(v_i, v_j) \). As \( v_j \) and \( v_k \) do not dominate each other, \( w(v_j) \leq w(v_k) + d_C(v_j, v_k) \). Since \( d_C(v_i, v_k) = d_C(v_i, v_j) + d_C(v_j, v_k) \), we have

\[
    w(v_i) \leq w(v_j) + d_C(v_i, v_j) \leq w(v_k) + d_C(v_j, v_k) + d_C(v_i, v_j) = w(v_k) + d_C(v_i, v_k),
\]

and

\[
    w(v_k) \leq w(v_j) + d_C(v_j, v_k) \leq w(v_i) + d_C(v_i, v_j) + d_C(v_j, v_k) = w(v_i) + d_C(v_i, v_k).
\]

Therefore, \( v_i \) and \( v_k \) do not dominate each other.

This proves the observation.

We are now in a position to prove the lemma.

**Proving the first lemma statement.** We start with the first lemma statement. Consider a vertex \( v_i \) of \( C \) that is not in \( S \). According to our algorithm, \( v_i \) may or may not be processed in the second stage. If \( v_i \) is processed in the second stage, then \( v_i \) was pushed into \( S \) during the first stage but is removed from \( S \) in the second stage (hence \( v_i \) was processed twice in the algorithm). If \( v_i \) is not processed in the second stage, then \( v_i \) was not in \( S \) at the end of the first stage (hence \( v_i \) is processed only once in the algorithm). In either case, \( v_i \) must be dominated by a vertex \( v_j \) that was in \( S \). If \( v_j \) is still in \( S \) at the end of the algorithm, then the first lemma statement is proved; otherwise, we can prove inductively that \( v_j \) is dominated by a vertex \( v_k \) in \( S \). By the above Observation, \( v_i \) is dominated by \( v_k \) and thus the first lemma statement follows.

**Proving the second lemma statement.** We next prove the second lemma statement. We first prove a claim: at any moment during the algorithm, for any two vertices \( v \) and \( u \) of \( S \) such that \( v \) is above \( u \) in the stack \( S \) and \( u \in C_{ccw}(v) \), \( v \) and \( u \) do not dominate each other.

We use mathematical induction to prove it, as follows.

The claim is vacuously true in the beginning of the algorithm because \( S = \emptyset \). We assume that the claim holds on \( S \) right before a vertex \( v_i \) is processed. We show below that the claim still holds on \( S \) after \( v_i \) is processed. We first consider the processing of \( v_i \) in the first stage of the algorithm. Let \( S \) refer to the stack right before \( v_i \) is processed. Let \( v \) be the top element of \( S \) if \( S \neq \emptyset \). According to our algorithm, \( v_i \) is pushed into \( S \) in the following four cases.

1. \( S = \emptyset \). In this case, \( S = \{v_j\} \) after \( v_i \) is processed. Hence, the claim trivially holds.
2. \( S \neq \emptyset \), and \( v \notin C_{ccw}(v_i) \). In this case, \( C_{ccw}(v_i) \cap S = \emptyset \), and thus the claim holds after \( v_i \) is processed.
3. \( S \neq \emptyset \), \( v \in C_{ccw}(v_i) \), and \( v \) and \( v_i \) do not dominate each other. Let \( v_j \) be any vertex of \( S \) such that \( v_j \in C_{ccw}(v_i) \). To prove the lemma, it suffices to show that \( v_j \) and \( v_i \) do not dominate each other.
Indeed, if $v_j = v$, then we know that $v$ and $v_i$ do not dominate each other. Otherwise, $v_j$ is below $v$. As $v_j \in CCW(v_i)$, since $v \in CCW(v_i)$, $v_j$ is also in $CCW(v)$. Since $v$ is above $v_j$ in $S$, by the induction hypothesis $v_j$ and $v$ do not dominate each other. Since both $v_j$ and $v$ are in $CCW(v_i)$, $d_C(v_i, v_j) = d_C(v_i, v) + d_C(v, v_j)$ holds. By the above Observation, $v_i$ and $v_j$ do not dominate each other. Hence, the claim holds after $v_i$ is processed.

4. $S \neq \emptyset$, $v \in CCW(v_i)$, and $v_i$ dominates $v$. In this case, $v_i$ is pushed into $S$ after some vertices including $v$ are popped out of $S$. First of all, since the claim holds on $S$, after vertices popped out of $S$, the claim still holds on the new $S$. Let $S$ refer to the stack right before $v_i$ is pushed in. Hence, the claim holds on $S$.

If $S = \emptyset$, then the claim still holds after $v_i$ is pushed in since $v_i$ will be the only vertex in $S$. Otherwise, let $v'$ be the top element of $S$. If $v' \notin CCW(v_i)$, then the claim still holds on $S$ after $v_i$ is pushed in. Otherwise, $v_i$ does not dominate $v'$ and both $v$ and $v'$ are in $CCW(v_i)$, and thus $v'$ is in $CCW(v)$. By the induction hypothesis, $v$ and $v'$ do not dominate each other. As $v_i$ dominates $v$, $v'$ cannot dominate $v_i$ since otherwise $v'$ would dominate $v$ by the above Observation. Hence, $v_i$ and $v'$ do not dominate each other. By the same argument as the above third case, $v_j$ and $v_i$ do not dominate each other for any vertex $v_j$ of $S$ with $v_j \in CCW(v_i)$. Hence, the claim holds after $v_i$ is pushed into $S$.

The above proves that the claim still holds after $v_i$ is processed in the first stage of the algorithm. Now consider processing $v_i$ in the second stage. If $v_i \notin CCW(v_i)$, then the algorithm stops without changing $S$ and thus the claim still holds on $S$. Otherwise, $v_i$ is removed from $S$, after which the claim still holds on $S$. Next, the algorithm processes $v_i$ in the same way as in the first stage and thus we can use the same argument as above to prove that the claim still holds after $v_i$ is processed. This proves the claim.

In the sequel we prove the second lemma statement by using the claim.

Consider two vertices $v_i$ and $v_j$ in $S$ at the end of the algorithm. Notice that either $v_i \in CCW(v_j)$ or $v_j \in CCW(v_i)$. Without loss of generality, we assume that the former case holds. Our goal is to show that $v_i$ and $v_j$ do not dominate each other.

If $v_j$ is above $v_i$ in $S$ at the end of the algorithm, then by the above claim, $v_j$ and $v_i$ do not dominate each other. Otherwise, according to our algorithm, $v_i$ must be last pushed into $S$ in the second stage while $v_j$ must be last pushed into $S$ in the first stage. Further, $v_i$ was also pushed into $S$ in the first stage before $v_j$ was processed and $v_i$ was never popped out of $S$ in the first stage. Hence, at the moment of the algorithm right before $v_j$ was processed, $v_i$ was already in $S$. At the moment of the algorithm right after $v_j$ was processed, $v_j$ was at the top of $S$ and $v_j$ was also in $S$. Since $v_i \in CCW(v_j)$, by the above claim, $v_j$ and $v_i$ do not dominate each other. This proves the second lemma statement.

### 2.3 Computing the diameter

In the following, we use $S$ to refer to the stack after the pruning algorithm. Note that $S$ cannot be empty. The following lemma shows how $S$ can help to find a vertex-weighted diametral pair of $C$.

**Lemma 5.** If $|S| = 1$, then any vertex-weighted diametral pair of $C$ must contain the only vertex in $S$. Otherwise, for any vertex $v$ of $C$ that is not in $S$, $v$ cannot be in any vertex-weighted diametral pair of $C$.

**Proof.** Suppose $|S| = 1$ and let $v$ be the only vertex in $S$. Then, by Lemma 4, every vertex of $C \setminus \{v\}$ is dominated by $v$. Let $(u^*, v^*)$ be a vertex-weighted diametral pair of $C$. At least one of $u^*$ and $v^*$ is not $v$. Without loss of generality, we assume $u^* \neq v$. Hence, $u^*$ is
dominated by $v$. Since $(u^*, v^*)$ is a vertex-weighted diametral pair, by Lemma 2 $v^*$ must be $v$. This proves the lemma for the case $|S| = 1$.

Now assume $|S| > 1$. Let $v$ be a vertex of $C$ that is not in $S$. By Lemma 4, $S$ has a vertex $u$ that dominates $v$. Assume to the contrary that $u$ is in a vertex-weighted diametral pair. Then by Lemma 2, the pair must be $(u, v)$. As $|S| > 1$, $S$ has another vertex $u'$ that is not in \{ $u$, $v$ \}. Since $u$ dominates $v$, we have $w(u) > w(v) + d_C(u, v)$. Since $u$ and $u'$ do not dominate each other, we have $w(u) \leq w(u') + d_C(u, u')$. Consequently, we can derive

$$w(u) + d_C(u, u') + w(u') \geq w(u) + w(u) > w(u) + d_C(u, v) + w(v).$$

But this incurs contradiction since $(u, v)$ is a vertex-weighted diametral pair.

In light of Lemma 5, if $|S| = 1$, we compute the diameter $\Delta(G)$ as follows. Let $v$ be the only vertex in $S$. We find the vertex $u \in C \setminus \{ v \}$ that maximizes the value $w(u) + d_C(u, v) + w(v)$, which can be done in $O(n)$ time with Lemma 3. By Lemma 5, $(u, v)$ is a vertex-weighted diametral pair and $\Delta(G) = w(u) + d_C(u, v) + w(v)$.

If $|S| > 1$, by Lemma 3, $\Delta(G) = \max_{u,v \in S}(w(u) + d_C(u, v) + w(v))$. The following lemma finds a vertex-weighted diametral pair and thus computes $\Delta(G)$ in linear time.

\textbf{Lemma 6.} A pair $(u, v)$ of vertices in $S$ that maximizes the value $w(u) + d_C(u, v) + w(v)$ can be found in $O(n)$ time.

\textbf{Proof.} Consider a vertex $u \in S$. Let $v_i$ and $v_j$ be two vertices in $S \cap C_{cw}(u)$ such that $d_C(u, v_i) < d_C(u, v_j)$ (e.g., see Fig. 2). Note that $d_C(u, v_j) = d_C(u, v_i) + d_C(v_i, v_j)$. We claim that $w(u) + d_C(u, v_i) + w(v_i) \leq w(u) + d_C(u, v_j) + w(v_j)$. Indeed, by Lemma 2, $v_i$ and $v_j$ do not dominate each other. Hence, $w(u) + d_C(u, v_i) + w(v_i) \leq w(u) + d_C(u, v_i) + w(v_j)$. The claim follows. The claim implies that if we consider the vertices $v$ of $S \cap C_{cw}(u)$ from $u$ along $C$ in clockwise order, then the value $w(u) + d_C(u, v) + w(v)$ is monotonically increasing. Similarly, if we consider the vertices $v$ of $S \cap C_{ccw}(u)$ from $u$ along $C$ in counterclockwise order, then the value $w(u) + d_C(u, v) + w(v)$ is monotonically increasing. Let $u^{cw}$ (resp., $u^{ccw}$) refer to the farthest vertex from $u$ in $C_{cw}(u)$ (resp., $C_{ccw}(u)$); e.g., see Fig. 2. Based on the above discussion, it holds that $\max_{v \in S \setminus \{u\}}(w(u) + d_C(u, v) + w(v)) = \max\{w(u) + d_C(u, u^{cw}) + w(u^{cw}), w(u) + d_C(u, u^{ccw}) + w(u^{ccw})\}$.

To find $u^{cw}$ and $u^{ccw}$, notice that if we traverse the vertices $u$ of $S$ along $C$ in clockwise order, then both $u^{cw}$ and $u^{ccw}$ are also ordered along $C$ in clockwise order. Hence, $u^{cw}$ and $u^{ccw}$ for all vertices $u$ of $S$ can be found in total $O(n)$ time by traversing the vertices of $S$.
10 Diameters of Unicycle Graphs and Diameter-Optimally Augmenting Trees

along C. Consequently, we have \( \max_{u \in S} \{w(u) + d_C(u, v) + w(v)\} = \max_{u \in S} \max \{w(u) + d_C(u, u^w) + w(u^w)\} \), which can be computed in \( O(n) \) time.

The proof of the following theorem summarizes our algorithm.

\( \textbf{Theorem 7.} \) The diameter (along with a diametral pair) of a unicycle graph can be computed in linear time.

\textbf{Proof.}\ Recall that there are two cases for a diametral pair \( (u^*, v^*) \) of \( G \): (1) both \( u^* \) and \( v^* \) are in the same tree of \( \mathcal{T}(G) \); (2) \( u^* \) and \( v^* \) are in two different trees of \( \mathcal{T}(G) \).

\( \text{The first case can be handled by computing the diameter and the corresponding diametral pair of each tree of } \mathcal{T}(G), \text{ which can be done in total } O(n) \text{ time. The longest diameter of these trees is kept as a candidate diameter of } G \text{ and the corresponding diametral pair is kept as a candidate diametral pair.} \)

\( \text{For the second case, we first compute the weights } w(v_i) \text{ for all vertices } v_i \text{ of the cycle } C; \text{ for each } v_i, \text{ we also store its farthest vertex } f(v_i) \text{ in } \mathcal{T}(v_i). \text{ Then, we perform the preprocessing of Lemma 3. Next, we run the pruning algorithm to obtain } S. \text{ If } |S| = 1, \text{ we find a vertex-weighted diametral pair of } C \text{ as described above; otherwise, we use Lemma 4 to find such a pair. In either case, let } (u, v) \text{ denote the pair. Then, a candidate diameter is } w(u) + d_C(u, v) + w(v). \text{ In addition, } (f(u), f(v)) \text{ is a candidate diametral pair of } G. \)

We compare the candidate diameters obtained from the above two cases and return the larger one as the diameter of \( G \); the corresponding diametral pair is a diametral pair of \( G \). The running time of the overall algorithm is \( O(n) \).

\( \textbf{3} \) \textbf{The Diameter-Optimally Augmenting Trees (DOAT)}

In this section, we solve the DOAT problem in \( O(n^2 \log n) \) time and \( O(n) \) space. Our algorithm for computing the diameter of a unicycle graph will be used as a subroutine.

\( \textbf{3.1 Observations} \)

We follow the same notation as in Section 2 such as \( \pi_G(s, t), d_G(s, t), \Delta(G) \).

Let \( T \) be a tree of \( n \) vertices such that each edge of \( T \) has a positive length. For any two vertices \( u \) and \( v \) of \( T \), we use \( e(u, v) \) to refer to the shortcut connecting \( u \) and \( v \); note that even if \( T \) already has an edge connecting them, we can always assume that there is an alternative shortcut (or we could also consider the shortcut as the edge itself with the same length). Let \( |e(u, v)| \) denote the length of \( e(u, v) \). Again, there is an oracle that can return the value \( |e(u, v)| \) in \( O(1) \) time for any shortcut \( e(u, v) \). Denote by \( T + e(u, v) \) the graph after adding \( e(u, v) \) to \( T \). The goal of the DOAT problem is to find a shortcut \( e(u, v) \) so that the diameter of the new graph \( \Delta(T + e(u, v)) \) is minimized. Let \( \Delta^*(T) \) be the diameter of an optimal solution. In the following we assume that \( \Delta^*(T) < \Delta(T) \), since otherwise any shortcut would be sufficient.

For any shortcut \( e(u, v) \), \( T \) has a unique path \( \pi_T(u, v) \) between \( u \) and \( v \). We make an assumption that \( |e(u, v)| < d_T(u, v) \) since otherwise \( e(u, v) \) can never be used (indeed, whenever \( e(u, v) \) was used in a shortest path, we could always replace it with \( \pi_T(u, v) \) to get a shorter path). This assumption is only for the argument of the correctness of our algorithm; the algorithm itself still uses the true value of \( |e(u, v)| \) (this does not affect the correctness,
because if $|e(u,v)| \geq d_T(u,v)$, then $e(u,v)$ cannot be an optimal shortcut. For the reference purpose, we refer to this assumption as the shortcut length assumption.

At the outset, we compute a diametral path $P$ of $T$ in $O(n)$ time. Let $v_1, v_2, \ldots, v_m$ be the vertices of $P$ ordered along it. Removing the edges of $P$ from $T$ results in $m$ connected components of $T$, each of which is a tree containing a vertex of $P$; we let $T(v_i)$ denote the tree containing $v_i$. For each $v_i$, we define a weight $w(v_i)$ as the distance from $v_i$ to its farthest vertex in $T(v_i)$. Let $T = \{T(v_i) | 1 \leq i \leq m\}$.

For any pair $(i, j)$ of indices with $1 \leq i < j \leq m$, we define a critical pair of vertices $(x,y)$ with $x \in T(v_i)$ and $y \in T(v_j)$ such that they minimize the value $d_{T(v_i)}(v_i, x') + |e(x', y)| + d_{T(v_j)}(y', v_j)$ among all vertex pairs $(x', y')$ with $x' \in T(v_i)$ and $y' \in T(v_j)$.

The following lemma will be used on several occasions later on.

**Lemma 8.** For any vertex $v$ in any tree $T(v_k) \in T$, it holds that $d_{T(v_k)}(v, v_k) \leq \min\{d_T(v_1, v_k), d_T(v_m, v_k)\}$. Also, $d_T(v_1, v_k) = d_P(v_1, v_k)$ and $d_T(v_m, v_k) = d_P(v_k, v_m)$.

**Proof.** Assume to the contrary that $d_{T(v_k)}(v, v_k) > \min\{d_T(v_1, v_k), d_T(v_m, v_k)\}$. Without loss of generality, we assume $d_{T(v_k)}(v, v_k) > d_T(v_1, v_k)$. Then, $d_T(v, v_m) = d_{T(v_k)}(v, v_k) + d_T(v_k, v_m) > d_T(v_1, v_k) + d_T(v_k, v_m) = d_T(v, v_m) = \Delta(T)$, a contradiction.

The second part of the lemma holds because $P$ is a path of $T$.

The following lemma demonstrates why critical pairs are “critical”.

**Lemma 9.** Suppose $e(u^*, v^*)$ is an optimal shortcut with $u^* \in T(v_i)$ and $v^* \in T(v_j)$. Then, $i \neq j$ and any critical pair of $(i, j)$ also defines an optimal shortcut.

**Proof.** Because $P$ is a diametral path of $T$, if $i = j$, then $P$ is still the shortest path from $v_1$ to $v_m$ in the new graph $T + e(u^*, v^*)$, and thus we have $\Delta^* = d_{T+e(u^*, v^*)}(v_1, v_m) = d_{T}(v_1, v_m) = \Delta(T)$. But this contradicts with our assumption $\Delta^* < \Delta(T)$. Hence, $i \neq j$.

Let $(x, y)$ be a critical pair of $(i, j)$ with $x \in T(v_i)$ and $y \in T(v_j)$. Define $\phi(v_i, v_j) = d_{T(v_i)}(v_i, x) + |e(x, y)| + d_{T(v_j)}(v_j, y)$ and $\phi^*(v_i, v_j) = d_{T(v_i)}(v_i, u^*) + |e(u^*, v^*)| + d_{T(v_j)}(v_j, v^*)$. By the definition of critical pairs, we have $\phi(v_i, v_j) \leq \phi^*(v_i, v_j)$. Also, due to the shortcut length assumption, it holds that $|e(v_i, v_j)| < d_T(v_i, v_j)$.

In the following, we prove that $d_{T+e(x,y)}(u, v) \leq \Delta(T + e(u^*, v^*))$ for any two vertices $u, v \in T$. This will prove the lemma. To simplify the notation, let $T_{xy} = T + e(x, y)$ and $T^* = T + e(u^*, v^*)$.

First of all, if $d_{T^*}(u, v) = d_{T^*}(u, v)$, then $d_{T_{xy}}(u, v) \leq d_T(u, v) = d_{T^*}(u, v) \leq \Delta(T^*)$. Below we assume $d_{T^*}(u, v) \neq d_{T^*}(u, v)$. Thus, $d_{T^*}(u, v) < d_T(u, v)$ and the shortest path $\pi^*(u, v)$ must contain the shortcut $e(u^*, v^*)$.

Let $T(v_k)$ and $T(v_h)$ be the trees of $T$ that contain $u$ and $v$, respectively, with $k \leq h$. Based on relationship of the indices $k$, $h$, $i$, and $j$, there are several cases.

1. $\{k, h\} \cap \{i, j\} = \emptyset$; e.g., see Fig. 3. In this case, we have

\[
d_{T_{xy}}(u, v) \leq d_T(u, v_i) + \phi(v_i, v_j) + d_{T}(v_j, v)
\leq d_T(u, v_i) + \phi^*(v_i, v_j) + d_{T}(v_j, v)
= d_{T^*}(u, v) \leq \Delta(T^*).
\]

Note that $d_T(u, v_i) + \phi^*(v_i, v_j) + d_{T}(v_j, v) = d_{T^*}(u, v)$ because $\pi^*(u, v)$ contains $e(u^*, v^*)$ and $k \leq h$. 


3. Figure 3 Illustrating the case \( \{k, h\} \cap \{i, j\} = \emptyset \).

4. Figure 4 Illustrating the case \( k = i \) and \( h = j \).

5. Figure 5 Illustrating the case \( k = i, h > i, \text{ and } h \neq j \).

6. Figure 6 Illustrating the case \( k = h = i \).

2. \( k = i \) and \( h = j \); e.g., see Fig. 4. In this case, we have

\[
\begin{align*}
d_{T,v}(u, v) &\leq d_{T(v_i)}(u, x) + |e(x, y)| + d_{T(v_j)}(y, v) \\
&\leq d_{T(v_i)}(u, v_i) + d_{T(v_j)}(v_i, x) + |e(x, y)| + d_{T(v_j)}(y, v_j) + d_{T(v_j)}(v_j, v).
\end{align*}
\]

By Lemma 8, \( d_{T(v_j)}(u, v_i) \leq d_T(v_1, v_i) \) and \( d_{T(v_j)}(v_j, v_i) \leq d_T(v_j, v_m) \). Hence, we can obtain

\[
\begin{align*}
d_{T,v}(u, v) &\leq d_T(v_1, v_i) + d_{T(v_j)}(v_i, x) + |e(x, y)| + d_{T(v_j)}(y, v_j) + d_T(v_j, v_m) \\
&= d_T(v_1, v_i) + \phi(v_i, v_j) + d_T(v_j, v_m) \\
&\leq d_T(v_1, v_i) + \phi^*(v_i, v_j) + d_T(v_j, v_m) \\
&= d_{T^*}(v_1, v_m) \leq \Delta(T^*).
\end{align*}
\]

3. \( k = i, h > i, \text{ and } h \neq j \); e.g., see Fig. 5. In this case, we have

\[
\begin{align*}
d_{T,v}(u, v) &\leq d_{T(v_i)}(u, x) + |e(x, y)| + d_{T(v_j)}(y, v_j) + d_T(v_j, v) \\
&\leq d_{T(v_i)}(u, v_i) + d_{T(v_j)}(v_i, x) + |e(x, y)| + d_{T(v_j)}(y, v_j) + d_T(v_j, v) \\
&\leq d_T(v_1, v_i) + d_{T(v_j)}(v_i, x) + |e(x, y)| + d_{T(v_j)}(y, v_j) + d_{T(v_j)}(v_j, v) \\
&= d_T(v_1, v_i) + \phi(v_i, v_j) + d_T(v_j, v) \\
&\leq d_T(v_1, v_i) + \phi^*(v_i, v_j) + d_T(v_j, v).
\end{align*}
\]

On the other hand, since \( h > i \), we also have

\[
\begin{align*}
d_{T,v}(u, v) &\leq d_{T(v_i)}(u, v_i) + d_T(v_i, v) \leq d_T(v_1, v_i) + d_T(v_i, v).
\end{align*}
\]

Notice that \( d_{T^*}(v_1, v) = \min\{d_T(v_1, v_i) + d_T(v_i, v), d_T(v_1, v_i) + \phi^*(v_i, v_j) + d_T(v_j, v)\} \). Combining the above two inequalities, we derive \( d_{T,v}(u, v) \leq d_{T^*}(v_1, v) \leq \Delta(T^*) \).

4. \( k < j, k \neq i, \text{ and } h = j \). This is case is symmetric to the above case and we can prove \( d_{T,v}(u, v) \leq \Delta(T^*) \) by a similar argument.
Figure 7 Illustrating the case $k < i$ and $h = i$.

5. $k = h = i$; e.g., see Fig. 7. We claim that this case cannot happen. Indeed, recall that $\pi_{T^*}(u, v)$ contains $e(u^*, v^*)$. Since both $u$ and $v$ are in $T(v_i)$, $\pi_{T^*}(u, v)$ must also contain $P[v_i, v_j]$, where $P[v_i, v_j]$ is the subpath of $P$ between $v_i$ and $v_j$. On the other hand, $\Delta(T) = d_T(v_i, v_j)$, which is equal to the length of $P$. Since $\Delta(T) > \Delta^* = \Delta(T^*)$, $\pi_{T^*}(v_i, v_m)$ must contain the shortcut $e(u^*, v^*)$ but cannot contain $P[v_i, v_j]$. This implies that $P[v_i, v_j]$ is not a shortest path between $v_i$ and $v_j$ in $T^*$. However, since $\pi_{T^*}(u, v)$, which is a shortest path between $u$ and $v$ in $T^*$, contains $P[v_i, v_j]$, $P[v_i, v_j]$ must be a shortest path between $v_i$ and $v_j$ in $T^*$. We thus obtain contradiction.

6. $k = h = j$. This case is symmetric to the above case. By a similar argument, we can show that it cannot happen.

7. $k < i$ and $h = i$; e.g., see Fig. 7. We claim that this case cannot happen. Indeed, since $\pi_{T^*}(u, v)$ contains $e(u^*, v^*)$ and $k < i$, $\pi_{T^*}(u, v)$ must also contain the subpath of $P$ between $v_i$ and $v_j$. By the same analysis as the above fifth case, we can obtain contradiction.

8. $k = j$ and $h > j$. This case is symmetric to the above case. By a similar argument, we can show that it cannot happen.

In summary, $d_{T_{xy}}(u, v) \leq \Delta(T^*)$ holds in all possible cases. The lemma thus follows.

### 3.2 Reducing DOAT to finding a shortcut for $P$

In light of Lemma 9, we reduce our DOAT problem on $T$ to finding a shortcut for the vertex-weighted path $P$ as follows.

For an index pair $(i, j)$ with $1 \leq i < j \leq m$, we define a shortcut $\tau(v_i, v_j)$ connecting $v_i$ and $v_j$ with length $|\tau(v_i, v_j)| = d_{T(v_i)}(v_i, x) + e(x, y) + d_{T(v_j)}(v_j, y)$, where $(x, y)$ is a critical pair of $(i, j)$. The diameter $\Delta(P + \tau(v_i, v_j))$ is defined as $\max_{1 \leq k < h \leq m} \{w(v_k) + d_{T^*}(v_k, v_h) + w(v_h)\}$. The diameter-optimally augmenting path (DOAP) problem on $P$ is to find a shortcut $\tau(v_i, v_j)$ so that the diameter $\Delta(P + \tau(v_i, v_j))$ is minimized; we use $\Delta^*(P)$ to denote the minimized diameter.

With the help of Lemma 9, the following lemma shows that the DOAT problem on $T$ can be reduced to the DOAP problem on $P$. Similar problem reductions were also used in [3][14].

**Lemma 10.** 1. For any index pair $(i, j)$ with $1 \leq i < j \leq m$, it holds that $\Delta(P + \tau(v_i, v_j)) = \Delta(T + e(x, y))$, where $(x, y)$ is a critical pair of $(i, j)$.
2. If $\tau(v_i, v_j)$ is an optimal shortcut for the DOAP problem on $P$, then $e(x, y)$ is an optimal shortcut for the DOAT problem on $T$, where $(x, y)$ is a critical pair of $(i, j)$.
3. $\Delta^*(T) = \Delta^*(P)$.

**Proof.** Before proving the first lemma statement, we first use it to prove the second and third lemma statements.
The second lemma statement. Suppose $\tau(v_i, v_j)$ is an optimal shortcut for the DOAP problem on $P$. By the first lemma statement, we have $\Delta(P + \tau(v_i, v_j)) = \Delta(T + e(x, y))$. Assume to the contrary that $e(x, y)$ is not an optimal shortcut for the DOAT problem for $T$, and let $e(u^*, v^*)$ instead be an optimal shortcut, with $u^* \in T(v_k)$ and $v^* \in T(v_h)$. Then, $\Delta(T + e(x, y)) > \Delta(T + e(u^*, v^*))$. By Lemma 9, $k \neq h$ and the critical pair $(x', y')$ of $(k, h)$ also defines an optimal shortcut for $T$. Hence, $\Delta(T + e(x, y)) > \Delta(T + e(x', y'))$. By the first lemma statement, $\Delta(P + \tau(v_k, v_h)) = \Delta(T + e(x', y'))$. Combining all above, we obtain

\[
\Delta(P + \tau(v_k, v_h)) = \Delta(T + e(x', y')) = \Delta(T + e(u^*, v^*)) < \Delta(T + e(x, y)) = \Delta(P + \tau(v_i, v_j)).
\]

But this incurs contradiction since $\tau(v_i, v_j)$ is an optimal shortcut for the DOAP problem on $P$. Hence, the second lemma statement is proved.

The third lemma statement. The third lemma statement follows immediately from the first two lemma statements. Indeed, suppose $\tau(v_i, v_j)$ is an optimal solution for the DOAP problem on $P$. Then, by the second lemma statement, $e(x, y)$ is an optimal shortcut for the DOAT problem on $T$, where $(x, y)$ is a critical pair of $(i, j)$. Hence, we have $\Delta(P^*) = \Delta(P + \tau(v_i, v_j))$ and $\Delta(T^*) = \Delta(T + e(x, y))$. By the first lemma statement, $\Delta(P + \tau(v_i, v_j)) = \Delta(T + e(x, y))$. Hence, $\Delta(T) = \Delta^*(P)$.

The first lemma statement. We now prove the first lemma statement. To simplify the notation, let $P' = P + \tau(v_i, v_j)$ and $T' = T + e(x, y)$. Our goal is to prove $\Delta(P') = \Delta(T')$.

First of all, because $(x, y)$ is a critical pair of $(i, j)$, $d_{P'}(v_k, v_h) = d_T(v_k, v_h)$ holds for any two vertices $v_k$ and $v_h$ of $P$ with $1 \leq k < h \leq m$.

We first prove that $\Delta(P') \leq \Delta(T')$. Let $v_k$ and $v_h$ be any two vertices of $P$ with $1 \leq k < h \leq m$. It suffices to show that $w(v_k) + d_{P'}(v_k, v_h) + w(v_h) \leq \Delta(T')$.

Let $u$ be the vertex of $T(v_k)$ farthest from $v_k$, i.e., $w(v_k) = d_{T(v_k)}(v_k, u)$. Similarly, let $v$ be the vertex of $T(v_h)$ farthest from $v_h$. Recall that $i < j$ and $k < h$. Depending on the values of $i, j, k, h$, there are several cases.

1. $i = k$ and $j = h$; e.g., see Fig. 8. In this case, $d_{T}(v_1, v_k) = d_{T}(v_1, v_k)$ and $d_{T}(v_h, v_m) = d_{T}(v_1, v_k)$. By Lemma 8, $d_{T(v_k)}(u, v_k) \leq d_{T(v_1)}(v_1, v_k)$ and $d_{T(v_h)}(v, v_h) \leq d_{T(v_h)}(v, v_m)$. Hence, we can derive

\[
\begin{align*}
 w(v_k) + d_{P'}(v_k, v_h) + w(v_h) &= d_{T(v_k)}(u, v_k) + d_{T(v_k, v_h)} + d_{T(v_h)}(v, v) \\
 &\leq d_{T}(v_1, v_k) + d_{T}(v_k, v_h) + d_{T}(v_h, v_m) \\
 &= d_{T}(v_1, v_k) + d_{T}(v_k, v_m) + d_{T}(v_m, v_h) \\
 &= d_{T}(v_1, v_m) \leq \Delta(T').
\end{align*}
\]

Figure 8 Illustrating the case $i, j \cap \{k, h\} = \emptyset$.

Figure 9 Illustrating the case $i, j \cap \{k, h\} = \emptyset$. 
2. $\{i, j\} \cap \{k, h\} = \emptyset$; e.g., see Fig. 10. In this case, the shortest path $\pi_{T'}(u, v)$ between $u$ and $v$ in $T'$ must contain $\pi_{T(v_h)}(u, v_k)$ and $\pi_{T(v_k)}(v_h, v)$. Hence, we have
\[
w(v_k) + d_{P'}(v_k, v_h) + w(v_h) = d_{T(v_k)}(u, v_k) + d_{T'}(v_k, v_h) + d_{T(v_h)}(v_h, v) = d_{T'}(u, v) \leq \Delta(T').
\]

3. $i = k$ and $j \neq h$; e.g., see Fig. 10. In this case, the shortest path $\pi_{T'}(v_1, v)$ between $v_1$ and $v$ in $T'$ must contain $\pi_{T}(v_1, v_k)$ and $\pi_{T(v_h)}(v_h, v)$. Hence, we have
\[
w(v_k) + d_{P'}(v_k, v_h) + w(v_h) = d_{T(v_k)}(u, v_k) + d_{T'}(v_k, v_h) + d_{T(v_h)}(v_h, v) \leq d_{T}(v_1, v_k) + d_{T'}(v_k, v_h) + d_{T(v_h)}(v_h, v) = d_{T'}(v_1, v) \leq \Delta(T').
\]

4. $i \neq k$ and $j = h$. This case is symmetric to the above case. By a similar argument, we can show that $w(v_k) + d_{P'}(v_k, v_h) + w(v_h) \leq \Delta(T')$.

5. $j = k$; e.g., see Fig. 11. In this case, we claim that $\pi_{T(v_h)}(u, v_k)$ is $\pi_{T'}(u, v_k)$. Indeed, assume to the contrary that this is not true. Then, $\pi_{T'}(u, v_k)$ must contain the shortcut $e(x, y)$. This further implies that $\pi_{T'}(u, v_k)$ must contain $\pi_T(v_1, v_k)$, which is the subpath of $P$ between $v_1$ and $v_k$ ($=v_j$). Since $(x, y)$ is a critical pair of $(i, j)$, $\pi_{T'}(v_i, v_k)$ is no shorter than the path $\pi' = \pi_{T(v_j)}(v_i, x) \cup e(x, y) \cup \pi_{T(v_j)}(y, v_k)$. Therefore, if we replace $\pi_{T}(v_i, v_k)$ by $\pi'$ in $\pi_{T'}(u, v_k)$, we can obtain another shortest path $\pi'_{T'}(u, v_k)$ from $u$ to $v_k$ in $T'$. However, $\pi'_{T'}(u, v_k)$ cannot be a shortest path as it contains $e(x, y)$ twice. This incurs contradiction and thus the claim follows.

Due to the claim, we have $d_{T'}(u, v) = d_{T(v_h)}(u, v_k) + d_{T'}(v_k, v_h) + d_{T(v_h)}(v_h, v)$. We can now obtain
\[
w(v_k) + d_{P'}(v_k, v_h) + w(v_h) = d_{T(v_k)}(u, v_k) + d_{T'}(v_k, v_h) + d_{T(v_h)}(v_h, v) = d_{T'}(u, v) \leq \Delta(T').
\]

6. $i = h$. This case is symmetric to the above case. By a similar argument, we can show that $w(v_k) + d_{P'}(v_k, v_h) + w(v_h) \leq \Delta(T')$.

This proves that $\Delta(P') \leq \Delta(T')$.

Next we prove that $\Delta(T') \leq \Delta(P')$. Let $u$ and $v$ be any two vertices of $T$ with $u \in T(v_k)$ and $v \in T(v_h)$, with $1 \leq k \leq h \leq m$. It suffices to show that $d_{T'}(u, v) \leq \Delta(P')$.

If $k \neq h$, then we can deduce
\[
d_{T'}(u, v) \leq d_{T(v_h)}(u, v_k) + d_{T'}(v_k, v_h) + d_{T(v_h)}(v_h, v) \leq w(v_k) + d_{T'}(v_k, v_h) + w(v_h) = w(v_k) + d_{P'}(v_k, v_h) + w(v_h) \leq \Delta(P').
\]
To find an optimal shortcut for the DOAP problem on $P$, for each $i \in [1, m-1]$, we will compute an index $j(i)$ that minimizes the diameter $\Delta(P + \pi(i, v_j))$ among all indices $j \in [i+1, m]$, i.e., $j(i) = \arg \min_{j=i+1}^{m} \Delta(P + \pi(i, v_j))$, as well as the diameter $\Delta(P + \pi(v_i, v_{j(i)}))$. After that, the optimal shortcut of $P$ is the one that minimizes $\Delta(P + \pi(v_i, v_{j(i)}))$ among the shortcuts $\pi(i, j(i))$ for all $i \in [1, m-1]$. We refer to the shortcuts for $\pi(v_i, v_j)$ for all $j \in [i+1, m]$ as $v_i$-shortcuts. Therefore, our goal is to find an optimal $v_i$-shortcut $\pi(i, j(i))$ for each $i \in [1, m-1]$. Let $n_i$ denote the number of vertices in $T(v_i)$, for each $1 \leq i \leq m$. Note that $n = \sum_{i=1}^{m} n_i$. Fix an index $i$ with $1 \leq i \leq m-1$. In the following, we will present an algorithm that computes an optimal $v_i$-shortcut $\pi(v_i, v_{j(i)})$ and the diameter $\Delta(P + \pi(v_i, v_{j(i)}))$ in $O(n \cdot n_i + n \log n)$ time and $O(n)$ space. In this way, solving the DOAP problem on $P$ takes $O(n^2 \log n)$ time and $O(n)$ space in total.

We introduce a domination relationship among $v_i$-shortcuts.
Lemma 13. The following lemma proves the correctness of the algorithm.

If \( \pi(v_i, v_j) \) dominates \( \pi(v_i, v_k) \), then shortcut \( \pi(v_i, v_k) \) can be ignored or "pruned".

\[
\Delta(P + \pi(v_i, v_j)) \leq \Delta(P + \pi(v_i, v_k)).
\]

Proof. Let \( u \) and \( v \) be any two vertices of \( P \). To prove the lemma, it suffices to show that \( d_{P+\pi(v_i, v_k)}(u, v) \leq d_{P+\pi(v_i, v_k)}(u, v) \).

If \( d_{P+\pi(v_i, v_k)}(u, v) = d_{P}(u, v) \), then \( d_{P+\pi(v_i, v_k)}(u, v) \leq d_{P}(u, v) = d_{P+\pi(v_i, v_k)}(u, v) \). Otherwise, the shortest path \( \pi_{P+\pi(v_i, v_k)}(u, v) \) contains the shortcut \( \pi(v_i, v_k) \). Hence, \( d_{P+\pi(v_i, v_k)}(u, v) \) is equal to either \( d_{P}(u, v) + |\pi(v_i, v_k)| + d_{P}(v_k, v) \) or \( d_{P}(u, v) + |\pi(v_i, v_k)| + d_{P}(v_e, v) \). We assume that it is the former case. Then we have

\[
\begin{align*}
& d_{P+\pi(v_i, v_k)}(u, v) = d_{P}(u, v) + |\pi(v_i, v_k)| + d_{P}(v_k, v) \\
& \geq d_{P}(u, v) + |\pi(v_i, v_j)| + d_{P}(v_j, v_k) + d_{P}(v_k, v) \\
& \geq d_{P}(u, v) + |\pi(v_i, v_j)| + d_{P}(v_j, v) \\
& \geq d_{P+\pi(v_i, v_j)}(u, v).
\end{align*}
\]

The lemma thus follows.

Let \( S_i \) be the set of all \( v_i \)-shortcuts, i.e., \( S_i = \{ \pi(v_i, v_j) \mid i + 1 \leq j \leq m \} \). In the following, we describe a pruning algorithm that computes a subset \( S_i \) of \( S_i \) such that no two shortcuts of \( S_i \) dominate each other and \( S_i \) contains at least one optimal \( v_i \)-shortcut. As will be seen later, these properties of \( S_i \) allow an efficient algorithm to find an optimal \( v_i \)-shortcut.

Before running the pruning algorithm, we compute the lengths of shortcuts of \( S_i \) by brute force, as follows. First, with \( O(n) \) time preprocessing, given any two vertices \( u \) and \( v \) with \( u \in T(v_i) \) and \( v \in T(v_j) \) for \( j \neq i \), we can compute \( d_T(u, v) \) in constant time. Consider a tree \( T(v_i) \) with \( j \geq i+1 \). Computing the length of \( \pi(v_i, v_j) \) reduces to finding a critical pair of \( (i, j) \). To this end, we compute \( d_T(v_i, u) + |e(u, v)| + d_T(v_e, v) \) for all vertices \( u \in T(v_i) \) and all vertices \( v \in T(v_j) \), which can be done in \( O(n_i \cdot n_j) \) time (and \( O(n) \) space).

As such, computing the lengths of all shortcuts of \( S_i \) takes \( O(n_i \cdot n) \) time.

Our pruning algorithm processes the shortcuts \( \pi(v_i, v_j) \) for all \( j = i+1, i+2, \ldots, m \) one by one. A stack \( S \) is maintained and \( S = \emptyset \) initially. Consider any \( j \in [i+1, m] \). If \( S = \emptyset \), then we push \( \pi(v_i, v_j) \) into \( S \). Otherwise, let \( \pi \) be the shortcut at the top of \( S \). If \( \pi \) and \( \pi(v_i, v_j) \) do not dominate each other, then we push \( \pi(v_i, v_j) \) into \( S \). Otherwise, if \( \pi \) dominates \( \pi(v_i, v_j) \), then we proceed on \( j+1 \), i.e., \( \pi(v_i, v_j) \) is pruned. If \( \pi(v_i, v_j) \) dominates \( \pi \), then we pop \( \pi \) out of \( S \) (i.e., \( \pi \) is pruned). Next, we keep popping the top element out of \( S \) until either \( S \) becomes \( \emptyset \) or \( \pi(v_i, v_j) \) does not dominate it; in either case we push \( \pi(v_i, v_j) \) into \( S \).

As the lengths of the shortcuts of \( S_i \) are available, the algorithm runs in \( O(n) \) time. The following lemma proves the correctness of the algorithm.

Lemma 13. After the algorithm, no two shortcuts of \( S_i \) dominate each other and \( S_i \) contains at least one optimal \( v_i \)-shortcut.

Proof. We first prove the following observation, which will be used for proving the lemma.

Observation. Consider any three indices \( j, k, \) and \( h \) with \( i < j < k < h \leq m \).

1. If \( \pi(v_i, v_j) \) dominates \( \pi(v_i, v_k) \), and \( \pi(v_i, v_k) \) dominates \( \pi(v_i, v_h) \), then \( \pi(v_i, v_j) \) dominates \( \pi(v_i, v_h) \).

2. If \( \pi(v_i, v_j) \) and \( \pi(v_i, v_k) \) do not dominate each other, and \( \pi(v_i, v_k) \) and \( \pi(v_i, v_h) \) do not dominate each other, then \( \pi(v_i, v_j) \) and \( \pi(v_i, v_h) \) do not dominate each other.
Proof of the observation. Since $i < j < k < h$, $dp(v_j, v_k) + dp(v_k, v_h) = dp(v_j, v_h)$.

1. If $\pi(v_i, v_j)$ dominates $\pi(v_i, v_k)$ and $\pi(v_i, v_k)$ dominates $\pi(v_i, v_j)$, then we have

$$\|\pi(v_i, v_j)\| + dp(v_j, v_k) = \|\pi(v_i, v_j)\| + dp(v_j, v_k) + dp(v_k, v_h) \leq \|\pi(v_i, v_k)\| + dp(v_k, v_h) \leq \|\pi(v_i, v_j)\|.$$ 

Hence, $\pi(v_i, v_j)$ dominates $\pi(v_i, v_k)$.

2. If $\pi(v_i, v_j)$ and $\pi(v_i, v_k)$ do not dominate each other, and $\pi(v_i, v_k)$ and $\pi(v_i, v_h)$ do not dominate each other, then we have

$$\|\pi(v_i, v_j)\| + dp(v_j, v_k) = \|\pi(v_i, v_j)\| + dp(v_j, v_k) + dp(v_k, v_h) \leq \|\pi(v_i, v_k)\| + dp(v_k, v_h) \leq \|\pi(v_i, v_j)\|.$$ 

Hence, $\pi(v_i, v_k)$ does not dominate $\pi(v_j, v_k)$.

Therefore, $\pi(v_i, v_j)$ and $\pi(v_i, v_h)$ do not dominate each other. 

This proves the observation.

We are now in a position to prove the lemma.

We first show that no two shortcuts in $S$ dominate each other by mathematical induction. Initially the statement holds, for $S = \emptyset$. We assume that the statement holds right before $\pi(v_i, v_j)$ is processed. Let $S$ refer to the stack right before $\pi(v_i, v_j)$ is processed; let $\pi$ be the shortcut at the top of $S$ if $S \neq \emptyset$. According to our algorithm, $\pi(v_i, v_j)$ is pushed into $S$ in the following cases.

1. $S = \emptyset$. In this case, $S$ has only one shortcut after $\pi(v_i, v_j)$ is pushed in. Hence, the statement trivially holds.

2. $S \neq \emptyset$, and $\pi$ and $\pi(v_i, v_j)$ do not dominate each other. Let $\pi'$ be any shortcut in $S$. To prove the statement holds on $S$ after $\pi(v_i, v_j)$ is pushed in, it suffices to show that $\pi'$ and $\pi(v_i, v_j)$ do not dominate each other. If $\pi' = \pi$, then this is obviously true. Otherwise, by the induction hypothesis, $\pi'$ and $\pi$ do not dominate each other. As $\pi$ and $\pi(v_i, v_j)$ do not dominate each other, by the above Observation, $\pi'$ and $\pi(v_i, v_j)$ do not dominate each other.

3. $S \neq \emptyset$, and $\pi(v_i, v_j)$ dominates $\pi$. In this case, $\pi$ is popped out of $S$, and afterwards, the algorithm keeps popping out the top element of $S$ until either $S$ becomes $\emptyset$ or $\pi(v_i, v_j)$ does not dominate the current top element of $S$, denoted by $\pi''$. In the former case, the statement trivially holds. In the latter case, we claim that $\pi''$ does not dominate $\pi(v_i, v_j)$. Indeed, by the induction hypothesis, $\pi$ and $\pi''$ do not dominate each other. Since $\pi(v_i, v_j)$ dominates $\pi$, $\pi''$ cannot dominate $\pi(v_i, v_j)$ since otherwise $\pi'$ would dominate $\pi$ by the above Observation. Then, following the same argument as the above case, $\pi(v_i, v_j)$ and $\pi'$ do not dominate each other for any $\pi'$ in the current stack $S$. The statement thus holds.

This proves that no two shortcuts in $S$ dominate each other. 

We next show that $S$ contains at least one optimal $v_i$-shortcut. Let $\pi^*$ be an optimal $v_i$-shortcut. If $\pi^*$ is in $S$, then we are done with the proof. Otherwise, according to our algorithm, $\pi^*$ must be dominated by another shortcut $\pi$ of $S$. By Lemma 12, $\pi$ is also an optimal $v_i$-shortcut. If $\pi$ is in $S$, then we are done with the proof. Otherwise, we can inductively show that $S$ contains an optimal $v_i$-shortcut.

This completes the proof of the lemma.
Lemma 14. An optimal $v_i$-shortcut in $S$ can be found in $O(n \log n)$ time and $O(n)$ space.

Proof. We first prove some properties that our algorithm relies on.

Consider the graph $P + \pi(v_i, v_j)$ for an index $j$ with $i < j \leq m$. By slightly abusing the notation, let $\Delta(i, j) = \Delta(P + \pi(v_i, v_j))$. Suppose $(v_a, v_b)$ is a diametral pair of $P + \pi(v_i, v_j)$ with $a < b$. Then, $\Delta(i, j) = w(v_a) + d_{P + \pi(v_i, v_j)}(v_a, v_b) + w(v_b)$.

We claim that if $a \in (1, i]$, then $(v_1, v_b)$ is also a diametral pair. Indeed, since $a \neq 1$, by Lemma 8 and the definition of $w(v_a)$, we have $w(v_a) \leq d_P(v_1, v_a)$. Hence, we can derive

$$w(v_a) + d_{P + \pi(v_i, v_j)}(v_a, v_b) + w(v_b) \leq d_P(v_1, v_a) + d_{P + \pi(v_i, v_j)}(v_a, v_b) + w(v_b)$$

$$= d_{P + \pi(v_i, v_j)}(v_1, v_b) + w(v_b)$$

$$\leq w(v_1) + d_{P + \pi(v_i, v_j)}(v_1, v_b) + w(v_b).$$

Hence, $(v_1, v_b)$ is also a diametral pair.

Similarly, we claim that if $b \in [j, m)$, then $(v_a, v_m)$ is also a diametral pair. The claim can be proved by a similar argument as above.

Note that since $a < b$, $a \neq m$ and $b \neq 1$. Due to the above two claims, we assume that $a \in \{1\} \cup (i, m)$ and $b \in (1, j) \cup \{m\}$. Based on the values of $a$ and $b$, we define the following five functions (e.g., see Fig. 14).

1. For the case $a = 1$ and $b = m$, we define

$$\alpha(i, j) = w(v_1) + d_P(v_1, v_i) + \langle \pi(v_i, v_j) \rangle + d_P(v_j, v_m) + w(v_m).$$

Hence, if $a = 1$ and $b = m$, we have $\Delta(i, j) = \alpha(i, j)$.

2. For the case $a = 1$ and $b \in (i, j)$, we define

$$\beta(i, j) = w(v_1) + \max_{i < b < j} \left\{ \min\{d_P(v_1, v_b), d_P(v_1, v_i) + \langle \pi(v_i, v_j) \rangle + d_P(v_j, v_b) + w(v_b)\} \right\}.$$
Hence, if \( a = 1 \) and \( b \in (i, j) \), we have \( \Delta(i, j) = \beta(i, j) \).

3. For the case \( a \in (i, m) \) and \( b = m \), we define

\[
\gamma(i, j) = \max_{i' < a < m} \left\{ w(v_{i'}) + d_P(v_{i'}, v_{i'}) + d_P(v_{i'}, v_{j}) + w(v_{j}) \right\}.
\]

Note that \( d_P(v_{i'}, v_{j}) = \min\{d_P(v_{i'}, v_m), d_P(v_m, v_{j})\} \) for each \( i' \in (i, j) \), and \( d_P(v_i, v_m) \) otherwise.

Hence, if \( a \in (i, m) \) and \( b = m \), we have \( \Delta(i, j) = \gamma(i, j) \).

4. For the case \( i < a < b < j \), we define

\[
\delta(i, j) = \max_{i' < a < b' < j} \left\{ w(v_{i'}) + \min\{d_P(v_{i'}, v_{b'}) + d_P(v_{i'}, v_{j}) + w(v_{j})\} \right\}.
\]

Hence, if \( a, b \in (i, j) \), we have \( \Delta(i, j) = \delta(i, j) \).

5. For the case \( a = 1 \) and \( b \in (1, i] \), we define

\[
\lambda(i, j) = \max_{1 < b' \leq i} \left\{ w(v_{b'}) + d_P(v_1, v_{b'}) + w(v_{b'}) \right\}.
\]

Hence, if \( a = 1 \) and \( b \in (1, i] \), we have \( \Delta(i, j) = \lambda(i, j) \).

With the above definitions, we have

\[
\Delta(i, j) = \max\{\alpha(i, j), \beta(i, j), \gamma(i, j), \delta(i, j), \lambda(i, j)\}.
\]

Hence, if \( j \) changes in \([i + 1, m]\), the graph of \( \Delta(i, j) \) is the upper envelope of the graphs of the five functions.

Recall that our goal is to find an optimal \( v_i \)-shortcut in \( S \). Let \( I \) denote the set of the indices \( j \) of all shortcuts \( \pi(v_i, v_j) \) of \( S \). We consider these indices of \( I \) in order. We intend to show that \( \Delta(i, j) \) is a unimodal function (first decreases and then increases) as \( j \) changes in \( I \). To this end, we prove that each of the above five functions is a monotonically increasing or decreasing function as \( j \) changes in \( I \). Note that each index of \( I \) is in \([i + 1, m]\). To simplify the notation, we simply let \( I = \{i + 1, i + 2, \ldots, m\} \), or equivalently, one may consider that our pruning algorithm does not prune any shortcut from \( S_i \) and thus \( S = S_i \).

As no two shortcuts of \( S \) dominate each other, \( \pi(v_i, v_j) \) and \( \pi(v_i, v_{j+1}) \) do not dominate each other for any \( j \in (i, m) \), i.e., \( [\pi(v_i, v_j)] + d_P(v_j, v_{j+1}) > [\pi(v_i, v_{j+1})] \) and \( [\pi(v_i, v_{j+1})] + d_P(v_j, v_{j+1}) > [\pi(v_i, v_j)] \). Relying on this property, we prove monotonicity properties of the five functions as follows. Consider any index \( j \in (i, m) \).

1. \( \alpha(i, j) > \alpha(i, j + 1) \).

   **Proof:**
   \[
   \begin{align*}
   \alpha(i, j + 1) &= w(v_{j+1}) + d_P(v_{j+1}, v_j) + d_P(v_j, v_{j+1}) + w(v_{j+1}) \\
   &< w(v_{j+1}) + d_P(v_{j+1}, v_j) + d_P(v_j, v_{j+1}) + d_P(v_{j+1}, v_m) + w(v_{j+1}) \\
   &= w(v_{j+1}) + d_P(v_{j+1}, v_j) + \pi(v_i, v_j) + d_P(v_j, v_{j+1}) + w(v_{j+1}) \\
   &= \alpha(i, j).
   \end{align*}
   \]

Hence, \( \alpha(i, j) \) is monotonically decreasing as \( j \) increases.

2. \( \beta(i, j) \leq \beta(i, j + 1) \).

   **Proof:** For each \( b' \in (i, j) \), we have
   \[
   \begin{align*}
   \min\{d_P(v_{b'}, v_{b'}) + d_P(v_{b'}, v_{j}) + d_P(v_{b'}, v_{j+1})\} &
   \leq \min\{d_P(v_{b'}, v_{j+1}) + d_P(v_{b'}, v_{j+1}) + d_P(v_{j+1}, v_{j+1})\} \\
   &= \min\{d_P(v_{b'}, v_{j+1}) + d_P(v_{j+1}, v_{j+1}) + d_P(v_{j+1}, v_{j+1})\}.
   \end{align*}
   \]

Hence, \( \beta(i, j) \leq \beta(i, j + 1) \) holds and \( \beta(i, j) \) is monotonically increasing as \( j \) increases.
3. $\gamma(i, j) \geq \gamma(i, j + 1)$.

Proof:
Let $a'$ be the index such that $\gamma(i, j + 1) = w(v_{a'}) + d_{P + \pi(v_{i}, v_{j+1})}(v_{a'}, v_{m}) + w(v_{m})$. Depending on whether $a' \in (i, j + 1)$ or $a' \in [j + 1, m)$, there are two cases.

- If $a' \in [j + 1, m)$, then
  \[ \gamma(i, j + 1) = w(v_{a'}) + d_{P + \pi(v_{i}, v_{j+1})}(v_{a'}, v_{m}) + w(v_{m}). \]
  
  On the other hand, $d_{P + \pi(v_{i}, v_{j})}(v_{a'}, v_{m}) = d_{P}(v_{a'}, v_{m})$. Hence, $\gamma(i, j) \geq w(v_{a'}) + d_{P}(v_{a'}, v_{m}) + w(v_{m}) = \gamma(i, j + 1)$.

- If $a' \in (i, j + 1)$, then
  \[ \gamma(i, j + 1) = w(v_{a'}) + \min\{d_{P}(v_{a'}, v_{m}), d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j+1})| + d_{P}(v_{j+1}, v_{m})\} + w(v_{m}). \]
  
  Since $\pi(v_{i}, v_{j})$ and $\pi(v_{i}, v_{j+1})$ do not dominate each other, we have
  \[ d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j+1})| + d_{P}(v_{j+1}, v_{m}) < d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j})| + d_{P}(v_{j}, v_{m}) \]
  \[ = d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j})| + d_{P}(v_{j}, v_{m}). \]

  Hence,
  \[ \gamma(i, j + 1) \leq w(v_{a'}) + \min\{d_{P}(v_{a'}, v_{m}), d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j})| + d_{P}(v_{j}, v_{m})\} + w(v_{m}). \]
  
  If $a' \in (i, j)$, then we have $\gamma(i, j) \geq w(v_{a'}) + \min\{d_{P}(v_{a'}, v_{m}), d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j})| + d_{P}(v_{j}, v_{m})\} + w(v_{m}) \geq \gamma(i, j + 1)$; otherwise, $\gamma(i, j) \geq w(v_{a'}) + d_{P}(v_{a'}, v_{m}) + w(v_{m}) \geq w(v_{a'}) + \min\{d_{P}(v_{a'}, v_{m}), d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j})| + d_{P}(v_{j}, v_{m})\} + w(v_{m}) \geq \gamma(i, j + 1)$.

  As such, in either case $\gamma(i, j) \geq \gamma(i, j + 1)$ holds.

Hence, $\gamma(i, j)$ is monotonically decreasing as $j$ increases.

4. $\delta(i, j) \leq \delta(i, j + 1)$.

Proof: For each $i < a' < b' < j$, we have,
\[ \min\{d_{P}(v_{a'}, v_{a'}'), d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j})| + d_{P}(v_{j}, v_{a'}')\} \]
\[ \leq \min\{d_{P}(v_{a'}, v_{a'}'), d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j+1})| + d_{P}(v_{j+1}, v_{a'}) + d_{P}(v_{j}, v_{a'}')\} \]
\[ \leq \min\{d_{P}(v_{a'}, v_{a'}'), d_{P}(v_{a'}, v_{i}) + |\pi(v_{i}, v_{j+1})| + d_{P}(v_{j+1}, v_{a'})\} \]

Hence, $\delta(i, j) \leq \delta(i, j + 1)$ and $\delta(i, j)$ is monotonically increasing as $j$ increases.

5. $\lambda(i, j) = \lambda(i, j + 1)$.

Proof: By definition, $\lambda(i, j)$ is constant for all indices $j$. Thus, $\lambda(i, j) = \lambda(i, j + 1)$.

On the basis of the above monotonicity properties of the functions, we present a binary search algorithm that finds an optimal $v_{i}$-shortcut in $O(n \log n)$ time and $O(n)$ space.

Our algorithm performs binary search on the indices $[l, r]$, with $l = i + 1$ and $r = m$ initially. In each step, we decide whether we will proceed on $[k, r]$ or on $[l, k]$, where $k = \left\lfloor \frac{l + r}{2} \right\rfloor$. To this end, we compute $\Delta(i, k)$ and $\Delta(i, k + 1)$. By Lemma 10, $\Delta(i, k) = \Delta(T + e(x, y))$ where $(x, y)$ is a critical pair of $(i, k)$. Since $T + e(x, y)$ is a unicycle graph, we compute $\Delta(T + e(x, y))$ in $O(n)$ time by Theorem 7. Therefore, $\Delta(i, k)$ can be computed in $O(n)$ time. Note that the algorithm of Theorem 7 also returns a diametral pair for $T + e(x, y)$, and we can decide which of the five cases for the functions $\alpha$, $\beta$, $\gamma$, $\delta$, and $\lambda$ the diametral pair belong to. We do the same for $\Delta(i, k + 1)$. Assume that $\Delta(i, k) = f(i, k)$ and $\Delta(i, k + 1) = g(i, k + 1)$, for two functions $f$ and $g$ in $\{\alpha, \beta, \gamma, \delta, \lambda\}$. Then we have the following cases
1. \( f = g \). We have the following subcases.
   - \( f = g \in \{ \beta, \delta \} \). In this case, our algorithm proceeds on the interval \([l, k]\). To see this, since both \( \beta \) and \( \delta \) are monotonically increasing functions, we have \( \Delta(i, j) \geq f(i, j) \geq f(i, k) = \Delta(i, k) \), for any \( j \in (k, r] \). As such, the diameter \( \Delta(i, j) \) would increase if we proceed on \( j \in (k, r] \).
   - \( f = g \in \{ \alpha, \gamma \} \). In this case, we proceed on the interval \([k, r]\) because both functions are monotonically decreasing.
   - \( f = g = \lambda \). In this case, we stop the algorithm and return \( T(i, k) \) as an optimal \( v_i \)-shortcut. To see this, \( \Delta(i, j) \geq \lambda(i, j) = \lambda(i, k) = \Delta(i, k) \) for any \( j \in [l, r] \). Hence, \( \Delta(i, j) \) achieves the minimum at \( j = k \) among all \( j \in [l, r] \).

2. \( f \neq g \). We have the following subcases.
   - One of \( f \) and \( g \) is \( \lambda \). In this case, by a similar argument as before, we return \( T(v_i, v_k) \) as an optimal \( v_i \)-shortcut if \( f = \lambda \), and return \( T(v_i, v_{k+1}) \) as an optimal \( v_i \)-shortcut if \( g = \lambda \).
   - \( \{f, g\} = \{ \beta, \delta \} \). In this case, since both \( \beta \) and \( \delta \) are increasing functions, by a similar argument as before, we proceed on the interval \([l, k]\).
   - \( \{f, g\} = \{ \alpha, \gamma \} \). In this case, since both \( \alpha \) and \( \gamma \) are decreasing functions, by a similar argument as before, we proceed on the interval \([k, r]\).
   - One of \( f \) and \( g \) is in \( \{ \beta, \delta \} \) and the other is in \( \{ \alpha, \gamma \} \). In this case, one of \( T(v_i, v_k) \) and \( T(v_i, v_{k+1}) \) is an optimal \( v_i \)-shortcut, which can be determined by comparing \( \Delta(i, k) \) with \( \Delta(i, k + 1) \). To see this, without loss of generality, we assume that \( f \in \{ \beta, \delta \} \) and \( g \in \{ \alpha, \gamma \} \). Hence, \( \Delta(i, j) \geq f(i, j) \geq f(i, k) = \Delta(i, k) \) for any \( j \in [k + 1, r] \), and \( \Delta(i, j) \geq g(i, j) \geq g(i, k + 1) = \Delta(i, k + 1) \) for any \( j \in [l, k] \). As such, \( \min\{\Delta(i, k), \Delta(i, k + 1)\} \leq \Delta(i, j) \) for all \( j \in [l, r] \).

The algorithm will find an optimal \( v_i \)-shortcut in \( O(\log n) \) iterations. As each iteration takes \( O(n) \) time, the total time of the algorithm is \( O(n \log n) \). The space is \( O(n) \).

The proof of the following theorem summarizes our algorithm for the DOAT problem.

**Theorem 15.** The DOAT problem on the tree \( T \) can be solved in \( O(n^2 \log n) \) time and \( O(n) \) space.

**Proof.** We first compute a diametral path \( P \) of \( T \) as well as the weights \( w(v_i) \) for all vertices \( v_i \) of \( P \), which takes \( O(n) \) time. Then, for each \( i \in [1, m - 1] \), we compute an optimal \( v_i \)-shortcut \( T(v_i, v_{j(i)}) \) and its diameter \( \Delta(P + T(v_i, v_{j(i)})) \) for the DOAT problem on the vertex-weighted path \( P \). To do so, we first compute the critical pairs of \((i, j)\) for all \( j \in [i + 1, m] \) and thus the lengths of the set \( S_i \) of the shortcuts \( T(v_i, v_j) \) for all \( j \in [i + 1, m] \). This step takes \( O(n \cdot n_i) \) time and \( O(n) \) space. Next, we run a pruning algorithm to prune those shortcuts that are dominated by others from \( S_i \) and obtain a subset \( S_i \) such that no two shortcuts of \( S_i \) dominate each other and \( S_i \) contains an optimal \( v_i \)-shortcut; this step takes \( O(n) \) time.

After having \( S_i \), by using the algorithm of Theorem 7 as a subroutine, a binary search algorithm can find an optimal \( v_i \)-shortcut \( T(v_i, v_{j(i)}) \) as well as the diameter \( \Delta(P + T(v_i, v_{j(i)})) \). Recall that the critical pairs of \((i, j)\) for all \( j \in [i + 1, m] \) have been computed. Let \((x_i, y_i)\) be the critical pair of \((i, j(i))\). We store \((x_i, y_i)\) and \( \Delta(P + T(v_i, v_{j(i)})) \). Other space used in this step will be disregarded when we run the same algorithm to compute the optimal \( v_i \)-shortcuts for other \( i \)'s; so the total space used in the algorithm is bounded by \( O(n) \). After the optimal \( v_i \)-shortcuts for all \( i \in [1, m - 1] \) are computed, we determine the index \( i \) with minimum \( \Delta(P + T(v_i, v_{j(i)})) \) and return \( c(x_i, y_i) \) as the optimal shortcut for the DOAT problem on \( T \) and return \( \Delta(P + T(i, j(i))) \) as the diameter. It takes \( O(n \cdot n_i + n \log n) \) time and \( O(n) \)
space to compute an optimal $v_i$-shortcut for each $i \in [1, m-1]$. Hence, computing optimal $v_i$-shortcuts for all $i \in [1, m-1]$ takes $O(n^2 \log n)$ time in total, for $\sum_{i=1}^{m} n_i = n$. The space is $O(n)$ because only constant space is occupied (for storing $(x_i, y_i)$ and $\Delta(P + \pi(v_i, v_{j(i)}))$) after an optimal $v_i$-shortcut is computed for each $i$.

References

1. N. Alon, A. Gyárfás, and M. Ruszinkó. Decreasing the diameter of bounded degree graphs. *Journal of Graph Theory*, 35:161–172, 2000.
2. S.W. Bae, M. de Berg, O. Cheong, J. Gudmundsson, and C. Levcopoulos. Shortcuts for the circle. *Computational Geometry: Theory and Applications*, 79:37–54, 2019.
3. D. Bilò. Almost optimal algorithms for diameter-optimally augmenting trees. In *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC)*, pages 40:1–40:13, 2018.
4. D. Bilò, Luciano Gualà, and Guido Proietti. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theoretical Computer Science*, 417:12–22, 2012.
5. J.-L. De Carufel, C. Grimm, A. Maheshwari, and M. Smid. Minimizing the continuous diameter when augmenting paths and cycles with shortcuts. In *Proceedings of the 15th Scandinavian Workshop on Algorithm Theory*, pages 27:1–27:14, 2016.
6. J.-L. De Carufel, C. Grimm, S. Schirra, and M. Smid. Minimizing the continuous diameter when augmenting a tree with a shortcut. In *Proceedings of the 15th Algorithms and Data Structures Symposium (WADS)*, pages 301–312, 2017.
7. V. Chepoi and Y. Vaxes. Augmenting trees to meet biconnectivity and diameter constraints. *Algorithmica*, 33(2):243–262, 2002.
8. F.R.K. Chung and M.R. Garey. Diameter bounds for altered graphs. *Journal of graph theory*, 8(4):511–534, 1984.
9. E.D. Demaine and M. Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *Proceedings of the 12th Scandinavian conference on Algorithm Theory (SWAT)*, pages 420–431, 2010.
10. A.M. Farley and A. Proskurowski. Computation of the center and diameter of outerplanar graphs. *Discrete Applied Mathematics*, 2:185–191, 1980.
11. G.N. Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.
12. F. Frati, S. Gaspar, J. Gudmundsson, and L. Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, 72:995–1010, 2015.
13. Y. Gao, D.R. Hare, and J. Nastos. The parametric complexity of graph diameter augmentation. *Discrete Applied Mathematics*, 161:1626–1631, 2013.
14. U. Große, J. Gudmundsson, C. Knauer, M. Smid, and F. Stehn. Fast algorithms for diameter-optimally augmenting paths. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming*, pages 678–688, 2015.
15. U. Große, J. Gudmundsson, C. Knauer, M. Smid, and F. Stehn. Fast algorithms for diameter-optimally augmenting paths and trees. *arXiv:1607.05547*, 2016.
16. T. Ishii. Augmenting outerplanar graphs to meet diameter requirements. *Journal of Graph Theory*, 74(4):392–416, 2013.
17. C. Johnson and H. Wang. A linear-time algorithm for radius-optimally augmenting paths in a metric space. In *Proceedings of the 16th Algorithms and Data Structures Symposium (WADS)*, pages 466–480, 2019.
18 C.-L. Li, S.T. McCormick, and D. Simchi-Levi. On the minimum-cardinality-bounded-diameter and the bounded-cardinality-minimum-diameter edge addition problems. *Operations Research Letters*, 11:303–308, 1992.

19 N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

20 E. Oh and H.-K. Ahn. A near-optimal algorithm for finding an optimal shortcut of a tree. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 59:1–59:12, 2016.

21 S. Olariu. A simple linear-time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics*, 34:121–128, 1990.

22 A.A. Schoone, H.L. Bodlaender, and J. Van Leeuwen. Diameter increase caused by edge deletion. *Journal of Graph Theory*, 11:409–427, 1997.

23 H. Wang. An improved algorithm for diameter-optimally augmenting paths in a metric space. *Computational Geometry: Theory and Applications*, 75:11–21, 2018.

24 H. Wang and Y. Zhao. A linear-time algorithm for discrete radius optimally augmenting paths in a metric space. In *Proceedings of the 32nd Canadian Conference on Computational Geometry (CCCG)*, pages 174–180, 2020.

25 R. Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47:1965–1985, 2018.