On Two Problems Regarding Farthest Distances in Continuous Networks

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

im Rahmen eines binationalen Promotionsverfahrens (cotutelle de thése)
gemeinsam angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg
sowie durch die Faculty of Graduate and Postdoctoral Affairs
der Carleton University in Ottawa, Ontario, Kanada

von Dipl.-Comp.-Math. Carsten Grimm
geboren am 11. März 1986 in Meerbusch

Betreuer

Prof. Dr. Stefan Schirra	Otto-von-Guericke-Universität Magdeburg
Prof. Dr. Michiel Smid	Carleton University

Gutachter

Prof. Dr. Stefan Schirra	Otto-von-Guericke-Universität Magdeburg
Prof. Dr. Michiel Smid	Carleton University
Prof. Dr. Christian Knauer	Universität Bayreuth

Magdeburg, den 21. Dezember 2017

Grimm, Carsten
On Two Problems Regarding Farthest Distances in Continuous Networks
Dissertation, Otto-von-Guericke-Universität Magdeburg
Magdeburg, Februar 2018.

Datum der Einreichung: 15. August 2017
Datum des Promotionskolloquiums: 21. Dezember 2017

DDC 511.5 Graphentheorie, 516 Geometrie, 518.1 Algorithmen

DOI `10.24352/UB.OVGU-2018-018`

Abstract

Consider the continuum of points along the edges of a network, i.e., a connected, undirected graph with positive edge weights. We measure the distance between these points in terms of the network distance, i.e., the weighted shortest path distance. The continuous diameter of a network is the largest network distance between any two points on the network.

We study two intertwined problems within this metric space: The first problem is to minimize the continuous diameter of a geometric network by introducing one or more shortcuts that may connect any two points along the network. The second problem is to develop efficient data structures that support queries for the farthest points from a query point along a network.

We study the first problem for geometric cycles and trees.

For geometric cycles, a single shortcut never decreases the continuous diameter and two shortcuts always suffice to reduce the continuous diameter—except when the cycle consists of two degenerate line segments. We characterize optimal pairs of shortcuts for both convex and non-convex cycles. This allows us to produce an optimal pair of shortcuts for a geometric cycle with straight-line edges that has n vertices and k reflex vertices in $O(n + k^2 n)$ time.

For geometric trees, we minimize the continuous diameter when adding a single shortcut. Unlike in the discrete version of this network augmentation problem, the continuous diameter may increase when we add a shortcut to a tree. We characterize for which trees a single shortcut is sufficient to reduce the continuous diameter of a tree. We develop an algorithm that produces an optimal shortcut for a geometric tree with n vertices in $O(n \log n)$ time.

For the second problem, we develop a sequence of data structures supporting farthest-point queries on increasingly complex types of networks. This sequence culminates in a data structure for series-parallel networks. A series-parallel network is a network where each bi-connected component can be generated from a single edge using series operations that introduce a new vertex along an existing edge and parallel operations that create a copy of an existing edge. These results are part of a larger strategy for treelike networks where we divide a network into its bi-connected components and combine data structures for each bi-connected component that are drawn from a catalogue of data structures for networks that we can handle efficiently.

Zusammenfassung

Wir betrachten das Kontinuum der Punkte entlang der Kanten eines Netzwerkes, d.h., eines zusammenhängenden, ungerichteten Graphen mit positiven Kantengewichten. Die Netzwerkdistanz zweier Punkte entlang eines Netzwerkes bezeichnet die Länge eines kürzesten gewichteten Pfades entlang des Netzwerkes. Der kontinuierliche Durchmesser eines Netzwerkes ist die größte Netzwerkdistanz von zwei Punkten auf dem Netzwerk.

Wir studieren zwei miteinander verflochtene Problemstellungen in diesem metrischen Raum: Das erste Problem besteht darin, die größte Netzwerkdistanz in einem geometrischen Netzwerk durch das Hinzufügen von Abkürzungen zu minimieren. Das zweite Problem besteht darin, effiziente Datenstrukturen zu entwickeln, mithilfe derer für einen Anfragepunkt die jeweils entferntesten Punkte entlang des Netzwerkes ermittelt werden können.

Wir studieren das erste Problem für in die Ebene eingebettete Polygone und Bäume.

Für jedes Polygon ist eine einzelne Abkürzung niemals ausreichend, um den kontinuierlichen Durchmesser zu verringern, wohingegen eine Reduktion des kontinuierlichen Durchmessers mit zwei Abkürzungen stets möglich ist, es sei denn, das Polygon besteht aus genau zwei kongruenten Strecken. Wir charakterisieren optimale Paare von Abkürzungen für konvexe wie nicht-konvexe Polygone. Hieraus leiten wir ein Verfahren her, welches ein optimales Paar von Abkürzungen für ein Polygon mit n Ecken und k Innenecken in $O(n + k^2n)$ Zeit bestimmt.

Für Bäume minimieren wir den kontinuierlichen Durchmesser mit einer einzelnen Abkürzung. Anders als bei der diskreten Version dieses Problems, kann der kontinuierliche Durchmesser bei Hinzufügen einer Abkürzung zu einem Baum ansteigen. Wir charakterisieren diejenigen Bäume, für die eine Abkürzung existiert, die den kontinuierlichen Durchmesser verringert. Ferner ermitteln wir eine optimale Abkürzung für einen Baum mit n Ecken in $O(n \log n)$ Zeit.

In Bezug auf das zweite Problem entwickeln wir eine Folge von Datenstrukturen für Anfragen zum Berichten der entferntesten Punkte in zunehmend komplexen Klassen von Netzwerken. Das Hauptresultat ist eine Datenstruktur für Series-Parallel-Netzwerke. Ein Series-Parallel-Netzwerk ist ein Netzwerk, dessen Zweifachzusammenhangkomponenten jeweils aus einer einzelnen initialen Kante durch eine Folge von Series-Operationen, welche einen Knoten auf einer Kante einfügen und Parallel-Operationen, welche eine bestehende Kante verdoppeln, hervorgehen. Diese Ergebnisse sind Bestandteil einer übergreifenden Strategie zur Unterstützung von Entfernteste-Punkte-Anfragen in baumartigen Netzwerken, bei der ein Netzwerk zunächst in seine Zweifachzusammenhangkomponenten zerlegt wird, um dann Datenstrukturen für die einzelnen Zweifachzusammenhangkomponenten, welche aus einem Katalog bereits bekannter Datenstrukturen für effizient handhabbare Netzwerktypen bezogen werden, zu kombinieren.

Acknowledgements

This thesis would not exist without the support and guidance of the many people who shaped my path throughout the last six years. I owe them a great debt of gratitude that I cannot express in words. Nevertheless, I will attempt to honour them in the following incomplete list.

I thank my parents Bärbel and Fred for their unconditional love and support.

I thank my academic parents Stefan Schirra and Michiel Smid for their wise counsel, their patience, as well as their tireless efforts to shepherd me to graduation. I thank Jean-Lou de Carufel, Anil Maheshwari, Vida Dujmović, and Pat Morin for their crucial role in the latter and for sharing their joy for Computational Geometry with me. I thank Prosenjit Bose for creating the unique research atmosphere at the Computational Geometry Lab and for imparting his hilarious stories; I am especially indebted to Jit for brokering the cotutelle agreement and, thereby, creating the exceptional opportunity to pursue my doctoral studies on two continents. I also thank everyone else who worked hard behind the scenes to make this arrangement possible. I thank Heidemarie Bräsel who put everything in motion by sending me on my first journey to Canada.

I thank the thesis committee for their kind verdict. I am thankful for the audience of my thesis defence that turned the final steps of my dissertation project into a rather pleasant experience. I thank Petra Specht, Martin Wilhelm, Marc Mörig, and everyone else who ensured that the graduation ceremony and the celebration after the defence were so cheerful and memorable—including even the part where I was dragged backwards on a horse through the city.

My special thanks go to Hazel Massay, who provided a home way from home, André van Renssen, Sander Verdonschot, Gehana Booth, Atchariya Fletcher, Anke Friederici, Christian Braune, Christoph Doell, Sebastian Gläß, Juliane Gläß, Claire Pennarun, Martin Wilhelm, and Christina Czaban. Without your friendship, I would not have arrived where I am now.

Contents

1 Introduction **1**
- 1.1 Preliminaries and Problem Definition . . . 2
  - 1.1.1 Minimizing the Diameter when Augmenting a Network with Shortcuts . 3
  - 1.1.2 Supporting Farthest-Point Queries . . . 4
  - 1.1.3 Series-Parallel Networks . . . 5
- 1.2 Related Work . . . 7
  - 1.2.1 Minimum-Diameter Network Augmentation . . . 7
  - 1.2.2 Network Voronoi Diagrams . . . 7
  - 1.2.3 Center Problems in Networks . . . 8
- 1.3 Structure and Results . . . 9

2 Cycle Shortcuts **11**
- 2.1 Useful Shortcuts for a Geometric Cycle . . . 12
- 2.2 Alternating vs. Consecutive . . . 15
- 2.3 Balancing Diametral Cycles . . . 19
- 2.4 Algorithm . . . 29
  - 2.4.1 Algorithm for Convex Cycles . . . 29
  - 2.4.2 Algorithm for Non-Convex Cycles . . . 39

3 Tree Shortcuts **41**
- 3.1 Usefulness . . . 42
- 3.2 Optimal Shortcuts . . . 47
- 3.3 Preparations for the Algorithm . . . 54
  - 3.3.1 Simplifying the Tree . . . 54
  - 3.3.2 States and Operations . . . 56
  - 3.3.3 Blocking . . . 58
  - 3.3.4 Sudden Optimality . . . 62
- 3.4 Continuous Algorithm . . . 65
  - 3.4.1 The Algorithm from the Pair State Perspective . . . 65
  - 3.4.2 Optimality . . . 67
- 3.5 Discretization . . . 69
  - 3.5.1 Simulating Phase I . . . 70
  - 3.5.2 Simulating Phase II . . . 71
  - 3.5.3 Simulating Phase III . . . 74

4 Data Structures for Farthest-Point Queries 79
4.1 Preliminaries 79
  4.1.1 Previous Work 81
  4.1.2 Structure and Results 89
4.2 Parallel-Path Networks 92
4.3 Bead-Chain Networks 96
  4.3.1 Queries from the Main Cycle 98
  4.3.2 Queries from the Good Arcs 103
  4.3.3 Queries from the Bad Arc 108
4.4 Abacus Networks 110
  4.4.1 Inward Queries 112
  4.4.2 Outward Queries 112
4.5 Two-Terminal Series-Parallel Networks 119
  4.5.1 The Nesting Hierarchy 119
  4.5.2 Local Query 125
  4.5.3 Downward Query 127
  4.5.4 Query Upward 132
4.6 Series-Parallel Networks 134

5 Conclusion 137
5.1 Commonalities 137
5.2 Potential Improvements 138
5.3 Future Research 139

Bibliography 141

1 Introduction

This thesis proposes a new setting for the study of certain graph problems by shifting from the discrete perspective, in terms of finitely many vertices, to a continuous perspective, in terms of an infinite continuum of points along the edges. We study two intertwined problems in this new setting: the search for shortcuts in a network that reduce the largest travel time along the network and a means to quickly report how far one can travel from a given location and where these farthest locations would be. By studying these problems in the continuous setting, we shine a new light on their counterparts from the discrete world, the minimum-diameter graph augmentation problem and the farthest-vertex network Voronoi diagram, respectively.

Shortcuts for Networks Imagine a network of subways or highways where we measure the distance between locations in terms of the travel time. An urban engineer might want to improve the worst-case travel time by introducing shortcuts. This work advises where these shortcuts should be built. For example, we show where to find the best shortcut for a subway network without loops and we show that a single shortcut never improves the worst-case travel time of a ring road. Our continuous perspective on network augmentation reflects that a shortcut road could connect any two locations along a highway.

Farthest-Point Queries Consider the task to find the ideal location for a new hospital within the network formed by the streets of a city. One criterion for this optimization would be the emergency unit response time, i.e., the worst-case time an emergency crew needs to drive from the hospital to the site of an accident. However, a location might be optimal in terms of emergency unit response time, but unacceptable with respect to another criterion such as construction costs. We provide a data structure that would allow a decision maker to quickly compare various locations in terms of emergency unit response time.

The selection of these two particular problems as the subject matter of this thesis is motivated mainly through two separate works by Aronov et al. [5] and by Große et al. [29].

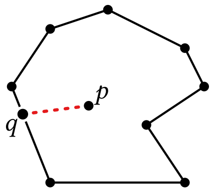


Figure 1.1: A feed-link that connects a point to a polygon.

Aronov et al. [5] seek to connect a point p in the plane to a polygonal cycle C using a straight-line segment pq, called a feed-link, from p to an anchor point q on C, as illustrated in Figure 1.1. The anchor point may be a vertex or a point along an edge. Aronov et al. [5] minimize the detour, i.e., the largest ratio of the largest geodesic distance between any two points along $C \cup pq$ and their Euclidean distance. Aronov et al. [5] motivate this type of feed-link problem by relating it to the task of connecting a new hospital to an existing system of roads. The choice of the detour as a target function reflects that the distances along the roads should approximate the distance "as the crow flies" as best as possible. Another target function that came to mind was the emergency unit response time, i.e., the largest travel time from the hospital to any other location. As it turns out, we minimize the emergency unit response time when connecting the hospital to a location where the farthest travel time within the existing network is minimal. As stated above, an ideal connection in terms of the emergency unit response time may be infeasible with respect to other aspects. Therefore, we became interested in supporting queries for farthest distances.

Große et al. [29] seek a shortcut for a polygonal path in the plane that connects two vertices of the path and minimizes the largest distance between any two vertices in the resulting network, as illustrated in Figure 1.2. Having studied networks in a continuous setting, this naturally generalizes to adding a shortcut to a polygonal path that connects any two points along the path and that minimizes the largest distance between any two points along the resulting network. After solving this problem for paths, we continue with cycles and trees.

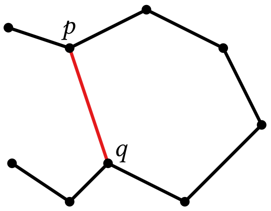
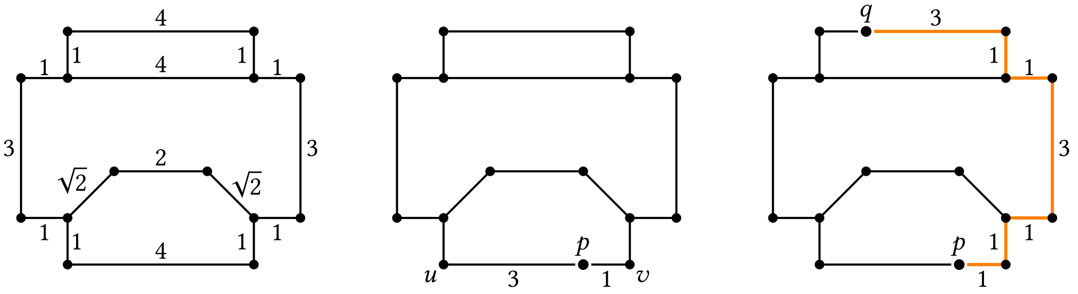


Figure 1.2: A shortcut for a path.

1.1 Preliminaries and Problem Definition

A *network* is a connected undirected graph $G = (V, E)$ with positive edge weights. We define points along the edges of a network as follows. Let $uv \in E$ be an edge in G with weight w_{uv} that connects the vertices $u, v \in V$. For every value $\lambda \in [0, 1]$, we define a point p on edge uv that subdivides uv into two sub-edges up and pv of weights $w_{up} = \lambda w_{uv}$ and $w_{pv} = (1 - \lambda) w_{uv}$, respectively. We write $p \in uv$ to indicate that p is a point along the edge uv, for some $\lambda \in [0, 1]$, and we write $p \in G$ to denote that p is a point along some edge of the network G.



(a) A network G.

(b) A point p along edge uv.

(c) A shortest path from p to q.

Figure 1.3: A geometric network G and a point p along edge uv of G that has relative position $\lambda = 3/4$ to u and v. We mark points along edges with discs sourrounded by a small gap to disinguish them from vertices marked by discs without any gap. A diametral pair of points $p, q \in G$ with network distance $d_G(p, q) = 11$, i.e., $\mathrm{diam}(G) = 11$.

A *geometric network* is a network that is embedded into the Euclidean plane and whose edges are rectifiable curves that are weighted with their Euclidean length, as illustrated in Figure 1.3. There is no ambiguity if two edges cross: there are two points along the network that correspond to the crossing in the plane, since points along edges are specified by their relative position to the endpoints of their containing edge, expressed by λ, and not by coordinates in the plane.

The *network distance* measures the distance between any two points $p, q \in G$—where p and q may be vertices or points along edges—in terms of the weighted length of a shortest path from p to q in G. The continuous diameter of G is the largest network distance between any two points on G, and it is denoted by $\operatorname{diam}(G)$, i.e., $\operatorname{diam}(G) = \max_{p,q \in G} d_G(p, q)$. In contrast, for a network G with vertex set V, the discrete diameter is the largest distance between any two vertices, i.e., $\max_{u,v \in V} d_G(u, v)$. We only consider the continuous diameter in this work. A pair $p, q \in G$ is *diametral* when its distance attains the continuous diameter, i.e., $diam(G) = d_G(p, q)$, and a *diametral path* in G is a shortest weighted path in G that connects a diametral pair of G.

The point c along a geometric tree T that minimizes the distance to the farthest leaf is called the *absolute center* of T. The absolute center is the midpoint of every diametral path of T. The intersection of all diametral paths of T is called the *backbone* of T, denoted by $\mathcal{B}$. The backbone $\mathcal{B}$ of T is either a path or a single vertex and $\mathcal{B}$ always contains the absolute center.

Let N be a network and let uv be an edge of N. Any pair of points $s, t \in uv$ defines a *sub-edge* st of uv of length $w_{st} = d_{uv}(s, t)$ that consists of the points along uv between s and t. A *sub-network* S of N is a network whose vertices are points along N and whose edges are sub-edges of N. A *cut vertex* in a network N is a vertex whose removal disconnects N. A *bi-connected component* of N is a maximal sub-network of N that does not contain any cut vertices.

Let P be a path in a network N whose endpoints may be vertices or points along the edges of N. The length of P, denoted by $|P|$, is the sum of the lengths of all edges and sub-edges of P. The length of a cycle C, denoted by $|C|$, is the sum of the lengths of the edges of C. For a point p along a cycle C, the *antipodal* of p is the farthest point $\bar{p}$ from p along C, i.e., $d_C(p, \bar{p}) = |C|/2$.

1.1.1 Minimizing the Diameter when Augmenting a Network with Shortcuts

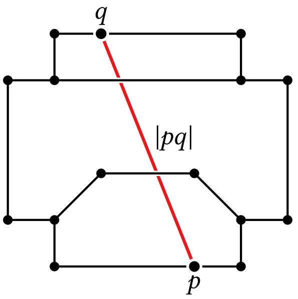


Figure 1.4: Augmenting a geometric network with a shorcut.

Let G be a geometric network. For any two points p and q along G that may be vertices or points along edges, the straight-line segment pq is called a *shortcut* for G. We augment a geometric network G with a shortcut pq, as illustrated in Figure 1.4. If they do not exist already, we introduce new vertices at p and at q, thereby subdividing the edges containing p and q. Then, we add the straight-line segment pq as a new edge to G where the edge weight of pq is its Euclidean length $|pq|$. We do not introduce any vertices at crossings between pq and any other edges of G; we can only enter and leave a shortcut at its endpoints. The resulting network is denoted by $G + pq$.

We aim to find a shortcut pq for G that minimizes the continuous diameter of the augmented network $G + pq$, i.e., we seek points $p, q \in G$ such that $\operatorname{diam}(G + pq) = \min_{r,s \in G} \operatorname{diam}(G + rs)$.

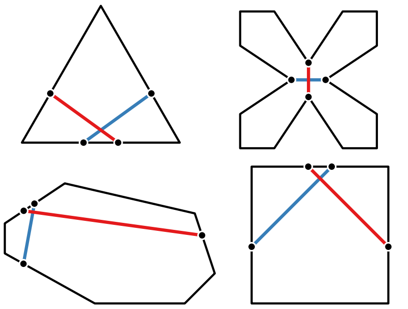


Figure 1.5: Four cycles with pairs of shortcuts that minimize the continuous diamter.

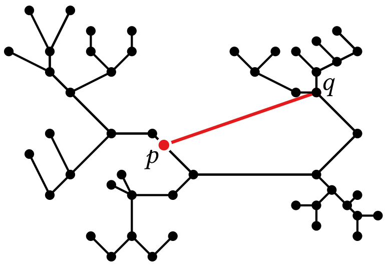
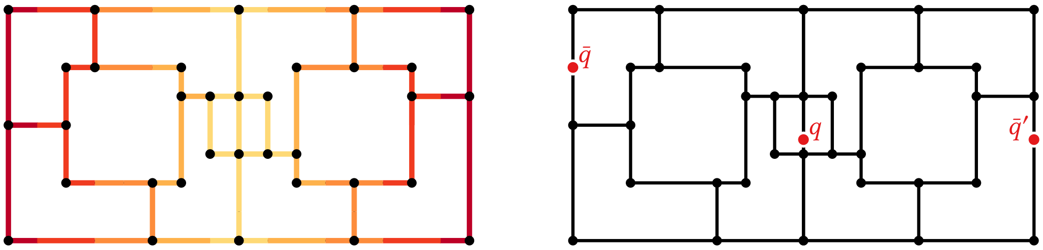


Figure 1.6: A geometric tree with a shortcut that minimizes the continuous diameter.

We study this problem for geometric trees and geometric cycles, as illustrated in Figures 1.5 and 1.6. A single shortcut cannot reduce the continuous diameter of a geometric cycle. Therefore, we seek a pair of shortcuts pq, rs for a cycle C that minimizes the continuous diameter of the augmented cycle $C + pq + rs$, i.e., $\mathrm{diam}(C + pq + rs) = \min_{a,b,c,d \in G} \mathrm{diam}(C + ab + cd)$. The endpoints of both shortcuts lie on the cycle C; we do not connect one shortcut to the other.

1.1.2 Supporting Farthest-Point Queries

We consider a network G. The *farthest distance* from a point $q \in G$ is the largest network distance between q and any other point along G. We denote the farthest distance from q by $\bar{d}_G(q)$, i.e., $\bar{d}_G(q) = \max_{p \in G} d_G(p, q)$. A *farthest point* from q is a point along G that attains the farthest distance from q, i.e., the set of farthest points from q is $F_G(q) = \{p \in G \mid \bar{d}_G(q) = d_G(p, q)\}$. As illustrated in Figure 1.7, these definitions of farthest distances and farthest points take all points along the network into account, i.e., vertices as well as points along edges.



(a) A heat map of the farthest distance.

(b) A farthest-point query with its answer.

Figure 1.7: Farthest-distance queries and farthest-point queries in a geometric network G.

Our goal is to develop a data structure for G that supports efficient queries for the farthest distance $\bar{d}_G(q)$ and the set of farthest points $F_G(q)$ from any query point $q \in G$. For a network with n vertices we aim for $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries, where $k = |F_G(q)|$ is the number of farthest points from q. We specify the query points as well as the reported farthest points by stating their containing edge and the relative position to the endpoints of that edge; this problem is independent of any embedding of G.

1.1.3 Series-Parallel Networks

In this work, we develop a data structure for farthest-distance queries and farthest-point queries on series-parallel networks. There are various conflicting definitions of two-terminal series-parallel networks and series-parallel networks. We rely on the definitions from the Information System on Graph Classes and their Inclusions [50, 51]. Please note that the two-terminal and series-parallel networks in this work are undirected.

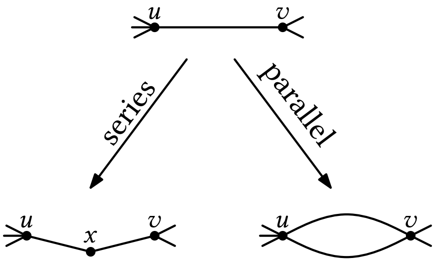


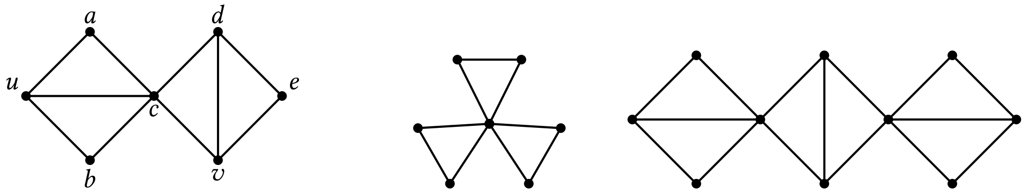
Figure 1.8: The operations that create series-parallel networks.

The term *series-parallel* stems from the two operations on the edges of a multigraph that are depicted in Figure 1.8. The *series operation* splits an existing edge uv into two new edges ux and xv where x is a new vertex. The *parallel operation* creates a copy of an existing edge. A graph G is *two-terminal series-parallel* if it can be generated from a single edge that connects two vertices u and v using a sequence of series and parallel operations. We call the vertices u and v the *terminals* of G and we say that G is two-terminal series-parallel with respect to u and v. The number λ of parallel operations that are required to generate G is called the *parallelism* of G.

When generating a two-terminal series-parallel graph, the intermediate networks may contain multiple edges. We assume, without loss of generality, that the final network is simple even if intermediate networks have multiple edges. We ensure this by applying additional series operations on edges that occur multiple times. A graph G is *series-parallel* when every bi-connected component B of G is *two-terminal series-parallel* with respect to some pair of vertices in B. A network is two-terminal series-parallel when its underlying graph is two-terminal series-parallel, and a network is series-parallel when its underlying graph is series-parallel. Figure 1.9 illustrates two-terminal series-parallel networks, networks that are series-parallel but not two-terminal series-parallel, and networks that are not series-parallel.

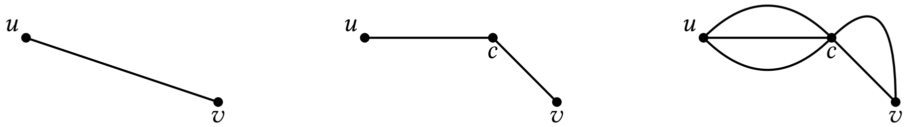
A *creation history* of a two-terminal series-parallel network N consists of a sequence of series operations and parallel operations that generate N. For a network with n vertices, we can check in $O(n)$ time whether N is two-terminal series-parallel by backtracking a creation history of N, i.e., reverting series operations and parallel operations in any order until no further operations can be reverted [20]. The network N is two-terminal series-parallel if and only if this process ends with a single edge. We can also determine in $O(n)$ time if a network N with n vertices is series-parallel by decomposing N into its bi-connected components [38] and then applying the recognition algorithm for two-terminal series-parallel networks on each bi-connected component.

There are several equivalent characterizations of series-parallel graphs: The class of series-parallel graphs, as defined above [51], is equivalent to the class of K_4-minor-free graphs [20], the class of partial 2-trees [59], and the class of graphs with tree-width two [11]. Every series-parallel network is planar [20] and every outer-planar network is series-parallel [20, 59]. Two-terminal series-parallel networks admit linear time solutions for several problems that are NP-hard on general networks [10, 55]. Since series-parallel networks have treewidth two [11], this applies to all problems with efficient algorithms on networks with bounded treewidth [4].



(a) A two-terminal network N.

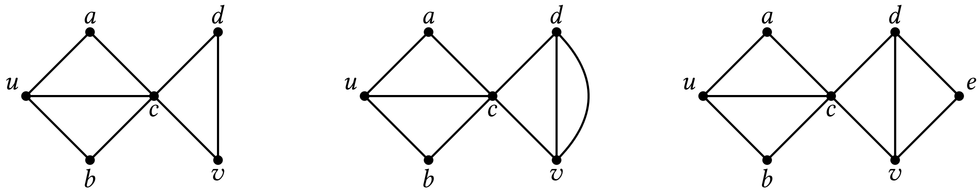
(b) Two series-parallel networks that are not two-terminal.



1. Start with a single edge.

2. Series Operation.

3. Parallel Operations.

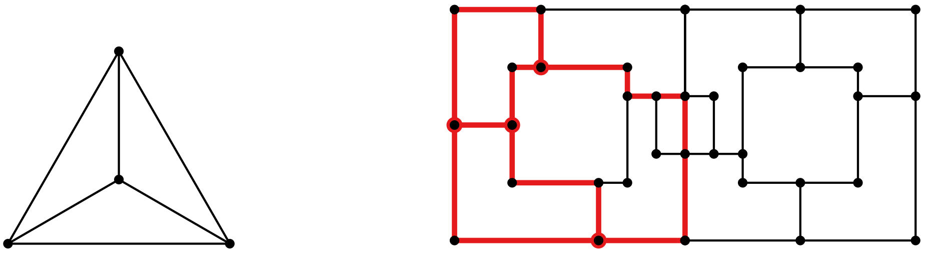


4. Series Operations.

5. Parallel Operation.

6. Series Operation.

(c) A creation history of the two-terminal series-parallel network N from Subfigure (a).



(d) The complete graph with four vertices K_4.

(e) A network with a K_4-minor.

Figure 1.9: The network N in Subfigure (a) is two-terminal series-parallel with respect to the vertices u and v, as certified by the creation history of N in Subfigure (c). On the other hand, N is not two-terminal series-parallel with respect to, e.g., the vertices a and b. The networks in Subfigure (b) are series-parallel, since their bi-connected components are two-terminal series-parallel, but they are not two-terminal series-parallel, because reverting all series-operations and all parallel operations does not yield a single edge. The complete graph with four vertices K_4 in Subfigure (d) is a forbidden minor for series-parallel networks. For instance, the network in Subfigure (e) cannot be series-parallel, since it contains a forbidden K_4-minor that is marked in red.

1.2 Related Work

We summarize the related literature on minimum-diameter network augmentation in various settings, Voronoi diagrams on networks, and center problems from location analysis.

1.2.1 Minimum-Diameter Network Augmentation

The problem of augmenting a network with shortcuts in order to reduce its farthest distance has been studied in various settings. Each setting differs in terms of the network at hand, the distances that are taken into account, the allowed endpoints of the shortcuts and their length. For instance, we distinguish whether the network is embedded into the plane (geometric) or whether it does not have a fixed embedding (abstract) and whether we consider distances between any points along the network (continuous) or only between vertices (discrete).

In the *abstract and discrete setting*, the goal is to minimize the discrete diameter of an abstract network with positive weights for the edges and non-edges by inserting non-edges as shortcuts. If the edges and non-edges have unit weight, then it is NP-hard to decide whether the diameter can be reduced below $D \geq 2$ by adding at most k shortcuts [19, 43, 52]. This problem remains NP-hard when the number of shortcuts is variable, even for trees [19]. On the other hand, Oh and Ahn [46] determine an optimal vertex-to-vertex shortcut that minimizes the discrete diameter of an abstract n-vertex tree with positive edge weights in $O(n^2 \log^3 n)$ time.

Minimum-diameter augmentation has also been studied as a bicriteria optimization in which both the diameter and the number (or cost) of the additional edges are minimized. For an overview on bicriteria approximation algorithms refer, for instance, to Frati et al. [24].

Große et al. [29] introduce the *geometric and discrete setting* in which the problem is to minimize the discrete diameter of a geometric network by connecting vertices with line segments. Große et al. [29] determine an optimal shortcut for a polygonal path with n vertices in $O(n \log^3 n)$ time using parametric search. Wang [60] recently improved this result to $O(n)$ time.

The stretch factor, i.e., the largest ratio of the network distance between any two vertices and their Euclidean distance, has also been studied as a target function [23, 44] in this setting.

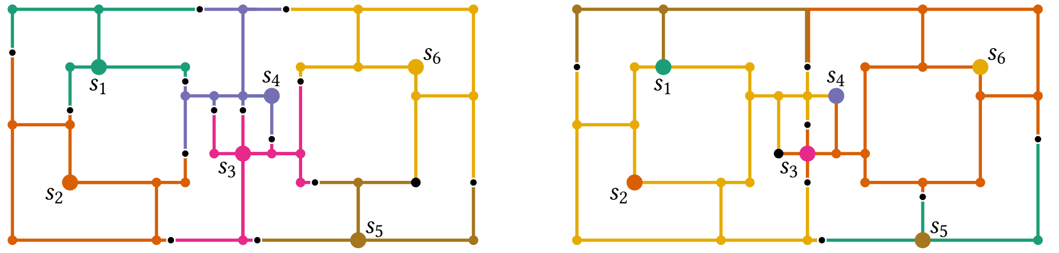
In the *geometric and continuous setting* [15], the task is to minimize the continuous diameter of a geometric network by inserting line segments that may connect any two points along the edges. We propose this setting as a natural generalization of the geometric discrete setting.

In the model studied in this work, a crossing of a shortcut with an edge or another shortcut is not a vertex: a path may only enter edges at their endpoints. In the *planar model* [14, 61], every crossing is a vertex of the resulting network, which leads to a different graph structure and, thus, continuous diameter. In the planar model, Yang [61] characterizes optimal shortcuts for a polygonal path. Cáceres et al. [14] determine in polynomial time whether the continuous diameter of a plane geometric network can be reduced with a single shortcut.

1.2.2 Network Voronoi Diagrams

Let G be a network and S a set of sites on G. A network Voronoi diagram subdivides G depending on which site in S is closest [33] or farthest [21, 48]. Okabe et al. [47] distinguish network Voronoi node diagrams, network Voronoi link diagrams, and network Voronoi area diagrams.

Let $G = (V, E)$ be a network and let $S \subseteq V$ be a set of vertices of G. The *network Voronoi node diagram* [47] or *graph Voronoi diagram* [21] partitions the vertices of G depending on which vertex among S is closest in terms of the network distance on G. Erwig [21] generalizes node diagrams to *inward* and *outward* graph Voronoi diagrams on directed networks.



(a) A nearest-site network Voronoi link diagram. (b) A farthest-site network Voronoi link diagram.

Figure 1.10: Two variants of the network Voronoi link diagram that subdivide a network depending on which site among the vertices $s_1, s_2, \dots, s_6$ is (a) closest or (b) farthest.

Consider a network G and a finite set S of points on G. The *network Voronoi link diagram* [33, 47] of S subdivides G into regions with common closest points in S with respect to the network distance, as illustrated in Figure 1.10. Variations of network Voronoi link diagrams include sites with additive or multiplicative weights, sites that are paths, and k-th nearest neighbour and farthest-point network Voronoi link diagrams [48]. Network Voronoi link diagrams have also been considered on directed networks [48], with coloured sites [39], and with moving sites [22].

Consider a plane network and a set of sites in the plane. Outside the network, we can move with unit speed; along the network, we can move faster depending on the respective edge weights. The *network Voronoi area diagram* [47] or *city Voronoi diagram* [3] subdivides the plane into regions depending on which site is closest with respect to this *travel time*. The research on network Voronoi area diagrams covers networks in the Euclidean plane [1, 8, 47], the L_1-plane [3], general metric planes [7], and farthest-point network Voronoi area diagrams [6].

The network Voronoi diagrams in the literature are defined with respect to a finite set of sites [47]. In contrast, we consider an implicit farthest-point network Voronoi diagram whose sites consist of the infinite continuum of all points on a network. In this sense, our research on farthest-point queries generalizes farthest-site network Voronoi link diagrams [12, 13].

1.2.3 Center Problems in Networks

Center problems revolve the task to locate a new position for some facility as a central position in a network, i.e., a position where the farthest distance is minimal. This area of research was sparked by the seminal work of Hakimi [32] on absolute centers.

We consider a graph $G = (V, E)$ with vertex weights w_v, for $v \in V$. As illustrated in Figure 1.11, an *absolute center* [32] of G is a point c along the edges of G that minimizes the largest weighted network distance from c to any of the vertices of the network G, i.e.,

$$\max_{v \in V} w_v d_G(c, v) = \min_{p \in G} \max_{v \in V} w_v d_G(p, v) \ .$$

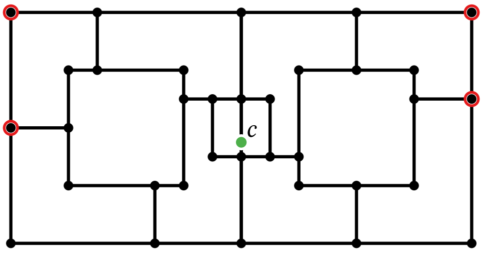


Figure 1.11: The absolute center c of a geometric network with unit vertex weights. The farthest vertices from c are highlighted with red circles.

When the vertex weights are non-negative, we can interpret the absolute center c as an ideal location for a new facility that serves clients that are situated at the vertices with non-zero weight. The vertex weights indicate the relative importance of the clients to a location analyst. Halpern and Maimon [34], Kincaid [41], Shi [53], Tansel [56], and Tansel, Francis, and Lowe [57, 58] survey results on the absolute center problem and its variants. ReVelle and Eiselt [49] summarize applications for center problems on networks.

A *continuous absolute center* is a point on a network with minimum farthest network distance. Ben-Moshe et al. [9] locate a continuous absolute center of a cactus with n vertices in $O(n)$ time. A network may have infinitely many continuous absolute centers, e.g., all points along a cycle are continuous absolute centers. Hansen, Labbé, and Nicolas [36] determine a representation of all continuous absolute centers on a network with n vertices and m edges in $O(m^2 \log n)$ time.

In the *absolute k-center problem* [40], we seek to place k *centers* $c_1, c_2, \ldots, c_k$ on a network $G = (V, E)$ minimizing the largest distance from any vertex $v \in V$ to its closest center, i.e.,

$$\max_{v \in V} \min_{i=1}^{k} d_G(v, c_i) = \min_{p_1, p_2, \ldots, p_k \in G} \max_{v \in V} \min_{i=1}^{k} d_G(v, p_i) \ .$$

In the *continuous absolute k-center problem*, we seek to place k centers that minimize the farthest distance from any point on G to the nearest center. Even though the absolute k-center problem and its continuous version are NP-hard in general [40, 45], some classes of networks, like trees, cactus networks, and almost trees admit efficient solutions [9, 25, 31, 35, 42, 53, 54].

In the *reverse center problem* [62] we are given a threshold $t > 0$ and ask for the smallest number p such that there is an absolute p-center where each point on the network is within network distance t of this absolute p-center. Zhang, Yang, and Cai [62] show that the reverse center problem is NP-hard and provide an approximation algorithm.

1.3 Structure and Results

In Chapter 2, we minimize the continuous diameter when augmenting a geometric cycle C with shortcuts. We discover that a single shortcut is never sufficient to reduce the continuous diameter, whereas two shortcuts always suffice, except in a degenerate corner case when C consists of two congruent line segments of length $|C|/2$. We characterize optimal pairs of shortcuts, which inform us how we can reduce the continuous diameter by sliding the endpoints p, q, r, s of a pair of shortcuts pq, rs along a cycle C, based on the cycles in $C + pq + rs$ that contain diametral pairs of the augmented cycle $C + pq + rs$. This allows us to determine an optimal pair of shortcuts for a convex polygonal cycle with n vertices in $O(n)$ time and an optimal pair of shortcuts for a non-convex polygonal cycle with n vertices and k reflex vertices in $O(k^2 n)$ time.

In Chapter 3, we minimize the continuous diameter when augmenting a geometric tree T. We observe that—unlike in the discrete version of this problem—adding a shortcut to a tree might increase the continuous diameter. We characterize the trees that have a useful shortcut, i.e., a shortcut that reduces the continuous diameter, as precisely those trees where the intersection of all diametral paths is neither a straight-line segment nor a point (i.e., a degenerate line segment). The intersection of all diametral paths, called the *backbone* $\mathcal{B}$ of T, plays a key role when locating an optimal shortcut. In the discrete setting, Große et al. [30] show that there exists an optimal shortcut for a tree with both endpoints along the backbone. We prove that this result carries over to the continuous setting and strengthen it: we show that every geometric tree has an optimal shortcut pq with both endpoints along the backbone such that the absolute center c of T lies on the path from p to q in T. This yields a restriction of the search space that allows us to find an optimal shortcut for a geometric tree with n straight-line edges in $O(n \log n)$ time. We develop this algorithm in two steps: First, we develop a set of rules that inform us how to continuously slide a shortcut along the backbone until we eventually reach an optimal shortcut. Then, we simulate this conceptual continuous algorithm with a discretization.

The structural results for augmenting trees and cycles with shortcuts hold for geometric networks whose edges are rectifiable curves, i.e., curves that have a well defined length. We describe the algorithmic results for geometric networks with straight-line edges; the techniques that we develop carry over to geometric networks with more general types of edges, e.g., edges that are algebraic curves of constant degree. Our model of computation is the real RAM.

In Chapter 4, we turn our attention to supporting farthest-distance queries and farthest-point queries in series-parallel networks. Our approach is to combine data structures for queries in less complex series-parallel networks in increasingly sophisticated ways to support queries in more complex series-parallel networks. This means we create data structures for a sequence of intermediate types of networks that themselves are build from data structures for cycles, trees, uni-cyclic networks, and cactus networks that I studied in my diploma thesis. Chapter 4 begins by summarizing our previous results, introducing the intermediate networks, and outlining their role as the stepping stones towards the data structure for series-parallel networks.

Chapters 2 to 4 are written to be self-contained and independent of each other and, thus, start by re-introducing the problem at hand. In Chapter 5, we point out some of the surprising commonalities between the different problems studied in this work, we discuss potential avenues for improving some of the results, and we outline directions for future research.

A preliminary version of the results regarding farthest-point queries in series-parallel networks from Chapter 4 has been presented at the 41st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2015) [28]. The results for convex cycles from Chapter 2 have been presented at the 5th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016) [15]. This publication also features an $O(n)$-time algorithm that minimizes the continuous diameter when adding a shortcut to a polygonal path with n vertices. The algorithm for polygonal paths is superseded by the algorithm for geometric trees with straight-line edges from Chapter 3, since the latter runs in $O(n)$ time for a path. The results from Chapter 3 regarding shortcuts for trees have been presented at the 15th International Symposium on Algorithms and Data Structures (WADS 2017) [16]; this contribution was invited to the special issue of Computational Geometry Theory and Applications for the best papers from WADS 2017.

2 Cycle Shortcuts

We seek to augment a cycle in the plane with shortcuts in order to minimize its continuous diameter. As it turns out, a single shortcut cannot reduce the continuous diameter of a cycle, yet two shortcuts suffice. Figure 2.1 illustrates examples of cycles with two optimal shortcuts.

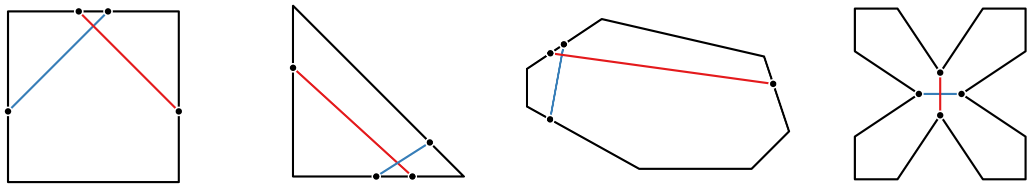


Figure 2.1: Examples for geometric cycles with optimal pairs of shortcuts.

We consider a geometric cycle C in the plane. For any two points $p, q \in C$, let $d_{\mathrm{ccw}}(p, q)$ and $d_{\mathrm{cw}}(p, q)$ be their counter-clockwise and clockwise distance along C, respectively. As illustrated in Figure 2.2, the network distance between p and q in C is $d(p, q) \coloneqq \min(d_{\mathrm{ccw}}(p, q), d_{\mathrm{cw}}(p, q))$. The continuous diameter of C is $\mathrm{diam}(C) = |C|/2$, where $|C|$ denotes the length of C. A geometric cycle C is *convex*, when C is the boundary of a convex set with non-empty interior. We require the interior of a convex cycle to be non-empty in order to exclude the *degenerate cycles* that consist of two congruent line segments of length $|C|/2$.

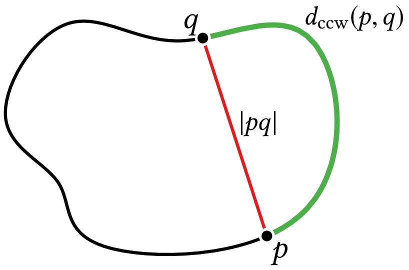


Figure 2.2: A geometric cycle with a shortcut pq such that $d(p, q) = d_{\mathrm{ccw}}(p, q)$.

We present the following structural and algorithmic results on optimal shortcuts for cycles.

1. A single shortcut cannot reduce the continuous diameter of a geometric cycle.
2. For every geometric cycle C, there exists a pair of shortcuts that reduces the continuous diameter—unless C is a degenerate cycle that consists of two congruent line segments.
3. For every geometric cycle C, there exists an optimal pair of shortcuts pq and rs such that their endpoints appear in alternating order p, r, q, s along C, as illustrated in Figure 2.1.
4. The largest simple cycles in the augmented cycle $C + pq + rs$ determine its continuous diameter. There are four candidates for *diametral cycles*, i.e., cycles containing diametral pairs: the two simple cycles in $C + pq + rs$ containing both shortcuts (⋈, ⧗), the larger simple cycle in $C + pq$ containing pq (⦸), and the larger simple cycle in $C + rs$ containing rs (⊘). By balancing these candidate cycles, we obtain our main structural results:

   a) Every geometric cycle C has a pair of optimal shortcuts pq, rs where the two simple cycles with both shortcuts are diametral ($\bowtie = ⧖ \geq ⦸, \oslash$), and where each of the larger simple cycles containing one shortcut ($⦸, \oslash$) is either diametral as well, or an endpoint of the corresponding shortcut lies at a reflex vertex of C.
   b) Every convex geometric cycle C has a pair of optimal shortcuts where all four of the candidate cycles are diametral and in balance ($\bowtie = ⧖ = ⦸ = \oslash$).
   c) A pair of shortcuts pq and rs, where the two simple cycles with both shortcuts are diametral ($\bowtie = ⧖ \geq ⦸, \oslash$), is optimal for a geometric cycle C when the sum of the lengths of the shortcuts $|pq| + |rs|$ is minimal among such pairs of shortcuts.

5. Based on these observations, we develop an algorithm that determines an optimal pair of shortcuts for a convex polygonal cycle with n vertices in $O(n)$ time.
6. We generalize our approach to non-convex cycles and determine an optimal pair of shortcuts for a geometric cycle C with n vertices and k reflex vertices in $O(k^2 n)$ time.

The remainder of this chapter is organized as follows. In Section 2.1, we establish the first two claims about the usefulness for one and two shortcuts for a cycle. In Section 2.2, we compare the two possible configurations for pairs of shortcuts and establish that the alternating configuration is better than the consecutive configuration. In Section 2.3, we characterize optimal shortcuts for convex and non-convex geometric cycles by balancing the diametral cycles. Using this characterization, we develop a linear-time algorithm that determines an optimal pair of shortcuts for convex cycles in Section 2.4.1. Building on the ideas of the algorithm for convex cycles, we conclude this chapter with an algorithm for non-convex cycles in Section 2.4.2.

2.1 Useful Shortcuts for a Geometric Cycle

We begin by showing that one shortcut is never sufficient to reduce the continuous diameter of a cycle and that two shortcuts always suffice for non-degenerate geometric cycles.

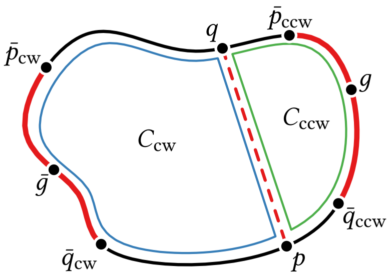


Figure 2.3: The unaffected regions (solid red) of a shortcut pq (dashed red) for a geometric cycle C. The point $\bar{x}_{\text{y}}$ denotes the farthest points from x along the cycle C_{y} for $x \in \{p, q\}$ and $\text{y} \in \{\text{cw}, \text{ccw}\}$. Any point g along the clockwise path from $\bar{p}_{\text{ccw}}$ to $\bar{q}_{\text{ccw}}$ has its farthest point $\bar{g}$ on the clockwise path from $\bar{q}_{\text{cw}}$ to $\bar{p}_{\text{cw}}$ and vice versa. The network distance between g and $\bar{g}$ is unaffected when we augment C with the shortcut pq.

Lemma 2.1. *Augmenting a geometric cycle C with a single shortcut pq has no effect on the continuous diameter, i.e., we observe* $\operatorname{diam}(C) = \operatorname{diam}(C + pq)$ *for every pair of points $p, q \in C$.*

Proof. Consider a shortcut pq for a geometric cycle C. Let C_{ccw} be the cycle consisting of pq and the counter-clockwise path from p to q along C, as illustrated in Figure 2.3. Let $\bar{p}_{\text{ccw}}$ and $\bar{q}_{\text{ccw}}$ be the farthest points from p and from q on C_{ccw}, respectively. Since $\bar{p}_{\text{ccw}}$ and $\bar{q}_{\text{ccw}}$ are antipodal from p and q in C_{ccw}, we have $d_C(\bar{q}_{\text{ccw}}, \bar{p}_{\text{ccw}}) = |pq|$ and $d_C(\bar{p}_{\text{ccw}}, q) = d_C(p, \bar{q}_{\text{ccw}})$.

Consider a point g along the clockwise path from $\bar{p}_{\text{ccw}}$ to $\bar{q}_{\text{ccw}}$ with $\bar{p}_{\text{ccw}} \neq g \neq \bar{q}_{\text{ccw}}$ and let $\bar{g} \in C$ be the farthest point form g with respect to C. The point $\bar{g}$ lies on the clockwise path from $\bar{q}_{\text{cw}}$ to $\bar{p}_{\text{cw}}$. We argue that augmenting C with pq does not affect the distance between g and $\bar{g}$.

We assume, for the sake of a contradiction, that augmenting C with the shortcut pq reduces the network distance between g and $\bar{g}$. This means that there is a shortest path from g to $\bar{g}$ in $C + pq$ that contains pq. Suppose this path reaches q before p. Then, the path from g to p via pq must be shorter than the path from g to p via C only, i.e., $d_{C+pq}(g, p) < d_C(g, p)$. However,

$$\begin{aligned}
d_{C+pq}(g,p) &= d_C(g,q) + |pq| && \text{(the path from g to $\bar{g}$ in $C+pq$ reaches q before p)}\\
&> d_C(p_{\text{ccw}}, q) + |pq| && \text{(g is on the path from $\bar{p}_{\text{ccw}}$ to $\bar{q}_{\text{ccw}}$ with $\bar{p}_{\text{ccw}} \neq g$)}\\
&= d_C(p, q_{\text{ccw}}) + |pq| && (d_C(p_{\text{ccw}}, q) = d_C(p, q_{\text{ccw}}))\\
&= d_C(p, q_{\text{ccw}}) + d_C(q_{\text{ccw}}, p_{\text{ccw}}) && (|pq| = d_C(q_{\text{ccw}}, p_{\text{ccw}}))\\
&> d_C(p, q_{\text{ccw}}) + d_C(q_{\text{ccw}}, g) && \text{(g is on the path from $\bar{p}_{\text{ccw}}$ to $\bar{q}_{\text{ccw}}$ with $g \neq \bar{q}_{\text{ccw}}$)}\\
&\geq d_C(p, g) \ .
\end{aligned}$$

Likewise, we derive a contradiction when we suppose that the shortest path from g to $\bar{g}$ in $C + pq$ contains pq and reaches p before q. Therefore, we have

$$\operatorname{diam}(C) = d_C(g, \bar{g}) = d_{C+pq}(g, \bar{g}) \leq \operatorname{diam}(C + pq) \ .$$

If $\operatorname{diam}(C) < \operatorname{diam}(C + pq)$, then $C + pq$ had a diametral pair u, v with $v \notin C$. This is impossible, as the farthest point $\bar{u}$ from u on C would then be farther from u than v, since $|pq| \leq d_C(p, q)$. □

As we have seen in the proof of Lemma 2.1, the farthest distance from some points on a geometric cycle C remains unchanged when we augment C with a single shortcut pq. The points that are unaffected by pq in this sense form the *unaffected region* of pq. The unaffected region of pq consists of the counter-clockwise path from $\bar{q}_{\text{ccw}}$ to $\bar{p}_{\text{ccw}}$ and the clockwise path from $\bar{q}_{\text{cw}}$ to $\bar{p}_{\text{cw}}$, as illustrated in Figure 2.3. Conversely, every point on C outside of the unaffected region of pq is affected by pq, i.e., uses pq on a shortest path to their farthest point in $C + pq$.

Consequently, we require at least two shortcuts to decrease the continuous diameter of a cycle. We call a pair of shortcuts pq and rs with $p, q, r, s \in C$ *useful* when $\operatorname{diam}(C) > \operatorname{diam}(C + rs + pq)$, and we call pq and rs *indifferent* otherwise, i.e., when $\operatorname{diam}(C) = \operatorname{diam}(C + pq + rs)$. A pair of shortcuts pq and rs is useful if and only if their unaffected regions are disjoint.

Consider a degenerate geometric cycle that consists of two congruent line segments of length $|C|/2$. No number of shortcuts can decrease the diameter of this degenerate cycle, since the endpoints of its line segment will always remain at distance $\operatorname{diam}(C) = |C|/2$. We argue that this is the only type of geometric cycle that does not possess a pair of useful shortcuts.

Theorem 2.2. *For every non-degenerate geometric cycle C, there exist three points $p, q, s \in C$ such that pq and qs form a useful pair of shortcuts for C.*

Proof. Let C be a geometric cycle. Suppose there exist three distinct points $p, q, s \in C$ with $d(p, q) = d(q, s) = |C|/4$ such that pq and qs satisfy $|pq| < d(p, q)$ and $|qs| < d(q, s)$.

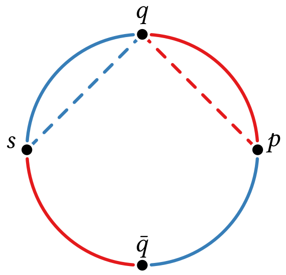
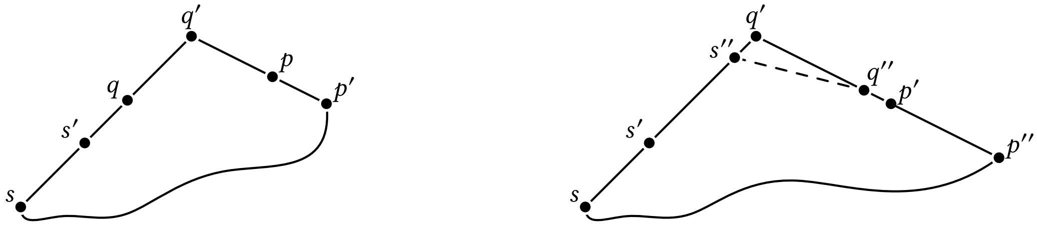


Figure 2.4: The unaffected regions of two touching shortcuts pq and qs. There is a small gap separating these regions from p, q, s, and $\bar{q}$, the antipodal of q.

We argue that the shortcuts pq and qs are useful for C. Let $\bar{q}$ denote the farthest point from q on C. As illustrated in Figure 2.4, the unaffected region of pq is confined to the interior of the clockwise paths from q to p and from $\bar{q}$ to s, and the unaffected region of qs is confined to the interior of the clockwise paths from s to q and from p to $\bar{q}$. Therefore, the unaffected regions of pq and qs are disjoint, i.e., pq and qs are useful shortcuts, i.e., $\operatorname{diam}(C) > \operatorname{diam}(C + pq + qs)$.

Suppose, on the other hand, that for every three points $p, q, s \in C$ with $d(p, q) = d(q, s) = |C|/4$ at least one of the line segments pq or qs is contained in C, i.e., $|pq| = d(p, q)$ or $|qs| = d(q, s)$. We prove that C is degenerate by showing that C contains a line segment of length $|C|/2$.



(a) Rotating from p, q, s to p', q', s'.

(b) Rotating from p', q', s' to p'', q'', s''

Figure 2.5: The impossible case of a non-degenerate cycle that contains at least one of the line segments sq or qp for every three points $p, q, s \in C$ with $d(s, q) = d(q, p) = |C|/4$.

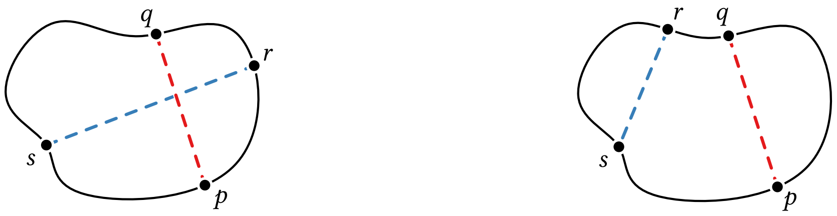
Let $p, q, s \in C$ be any three points with $d(p, q) = d(q, s) = |C|/4$. Then, C contains pq or qs. We assume, without loss of generality, that C contains the line segment qs. Otherwise, we swap p and s. We move p, q, and s clockwise along C while maintaining $d(p, q) = d(q, s) = |C|/4$ until we arrive at the first positions p', q', and s', where C contains $q'p'$. This means C contains both sq' and $q'p'$, as illustrated in Figure 2.5a. The points p', q' and s' exist and satisfy $0 \leq |sq'| \leq |C|/2$, since C cannot contain a line segment that is longer than $|C|/2$. If $|sq'| = |C|/2$, then the clockwise path from s to q' along C is a line segment of length $|C|/2$ and, thus, C is degenerate.

Suppose $|sq'| < |C|/2$. We move the three points p', q', and s' clockwise by a distance of $|C|/4 - \epsilon$ for some ϵ with $0 < \epsilon < |C|/4$. Let p'', q'', and s'' be the resulting points. As illustrated in Figure 2.5b, s'' lies on $s'q'$ at distance ϵ from q', and q'' lies on $q'p'$ at distance ϵ from p'. Since $d(s'', q'') = d(q'', p'') = |C|/4$, the cycle C contains $s''q''$ or $q''p''$. However, $s''q''$ cannot be contained in C, since otherwise q' would be contained in the line segment $s''q''$. This would contradict the choice of q' as the first point where $q'p'$ is contained in C. Therefore, C contains the line segment $q''p''$ whose length is $|q'p''| = |q'p'| + |p'p''| = |C|/2 - \epsilon$. Since this argument holds for arbitrary small values of ϵ, the cycle C contains a line segment of length $|C|/2$.

In summary, if a geometric cycle possesses three points $p, q, s \in C$ with $d(p, q) = d(q, s) = |C|/4$ and $|pq| < d(p, q)$ and $|qs| < d(q, s)$, then the shortcuts pq and qs are useful for C. On the other hand, if C does not possess three such points, then C must be a degenerate cycle that consists of two line segments of length $|C|/2$. Therefore, every geometric cycle that is not a degenerate cycle consisting of two line segments does possess a pair of useful shortcuts. □

2.2 Alternating vs. Consecutive

When placing two shortcuts pq and rs on a cycle C, we distinguish whether their endpoints appear in alternating order or in consecutive order along the cycle, as illustrated in Figure 2.6. We show that there is always an optimal pair of shortcuts in the alternating configuration.



(a) Alternating configuration.

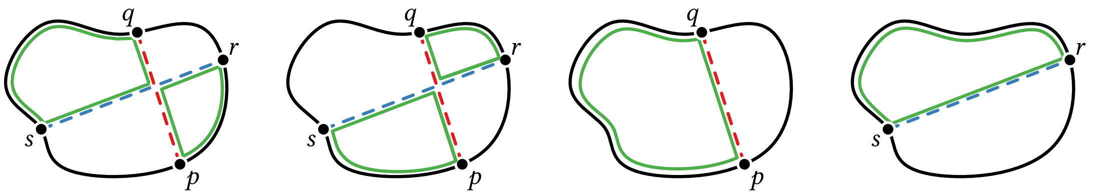
(b) Consecutive configuration.

Figure 2.6: The two cases for adding two shortcuts pq and rs to a cycle C. The endpoints of the shortcuts appear in alternating order p, r, q, and s, as shown in (a), or in consecutive order p, q, r, and s, as shown in (b). The two cases overlap when q coincides with r.

We call a cycle in the augmented cycle $C + pq + rs$ *diametral* when it contains a diametral pair. Each configuration has five candidates for diametral cycles: two that use both shortcuts, two that use one of the shortcuts, and one (C) that does not use any shortcut. Figures 2.7 and 2.8 illustrate the candidates for diametral cycles in each configuration, except for the cycle C itself.

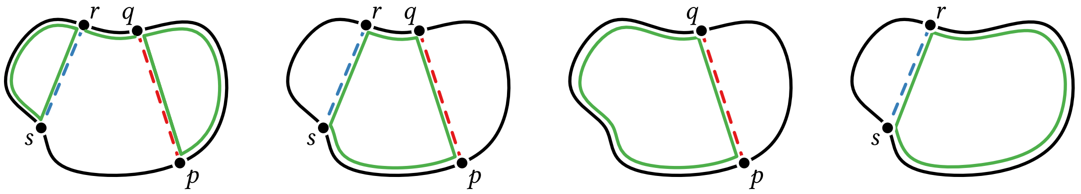
We distinguish the cycles that contain only one of the two shortcuts as follows. We color pq red and rs blue and we refer to the larger simple cycle in $C + pq$ that contains pq as the *red split cycle* and we refer to the larger simple cycle in $C + rs$ that contains rs as the *blue split cycle*.

For consecutive shortcuts, notice that even though the handset is not a simple cycle, it might still contain a diametral pair. Moreover, the base station is only listed for completeness: it is never larger than the red split cycle and the blue split cycle and, therefore, never diametral.



(a) The bowtie (⋈). (b) The hourglass (⧖). (c) Red split cycle (⦸). (d) Blue split cycle (⊘).

Figure 2.7: The candidates for diametral cycles, except C, in the alternating configuration.



(a) The handset. (b) The base station. (c) Red split cycle. (d) Blue split cycle.

Figure 2.8: The candidates for diametral cycles, except C, in the consecutive configuration.

For the following, let the points $\bar{x}_y$ with $x \in \{p, q, r, s\}$ and $y \in \{\text{cw}, \text{ccw}\}$ be defined as in Figure 2.3, e.g., let $\bar{r}_{\text{cw}}$ be the farthest point from r on the cycle $C_{\text{cw}}(r, s)$ that consists of rs and the clockwise path from r to s along C, and let $\bar{q}_{\text{ccw}}$ be the farthest point from q on the cycle $C_{\text{ccw}}(p, q)$ that consists of pq and the counter-clockwise path from p to q.

Lemma 2.3. *Two alternating shortcuts pq and rs are useful for a cycle C if and only if $|pq| < d(p, q)$ and $|rs| < d(r, s)$ and $|pq| + |rs| < d_{\text{ccw}}(r, q) + d_{\text{ccw}}(s, p)$ and $|pq| + |rs| < d_{\text{ccw}}(p, r) + d_{\text{ccw}}(q, s)$.*

Proof. Suppose pq and rs are useless alternating shortcuts with $|pq| < d(p, q)$ and $|rs| < d(r, s)$.

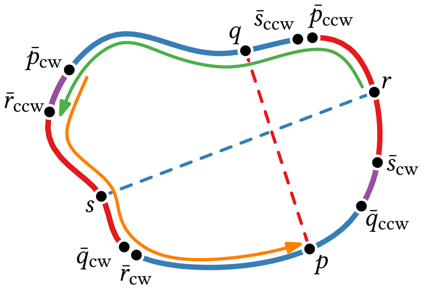


Figure 2.9: Overlap on the bowtie.

This means the unaffected regions of pq and rs overlap along the bowtie or along the hourglass. Since these cases are symmetric, we consider only the former in the following.

Suppose the unaffected regions of pq and rs overlap along the bowtie. As illustrated in Figure 2.9, this overlap consists of the clockwise path from $\bar{r}_{\text{ccw}}$ to $\bar{p}_{\text{cw}}$ and, symmetrically, of the clockwise path from $\bar{s}_{\text{cw}}$ to $\bar{q}_{\text{ccw}}$. The sum of the lengths of the counter-clockwise paths from r to $\bar{r}_{\text{ccw}}$ and from $\bar{p}_{\text{cw}}$ to p is at least the length of the counter-clockwise path from r to p, i.e., $d_{\text{ccw}}(\bar{p}_{\text{cw}}, p) + d_{\text{ccw}}(r, \bar{r}_{\text{ccw}}) \geq d_{\text{ccw}}(r, p)$. This is equivalent to $|pq| + |rs| \geq d_{\text{ccw}}(r, q) + d_{\text{ccw}}(s, p)$, because

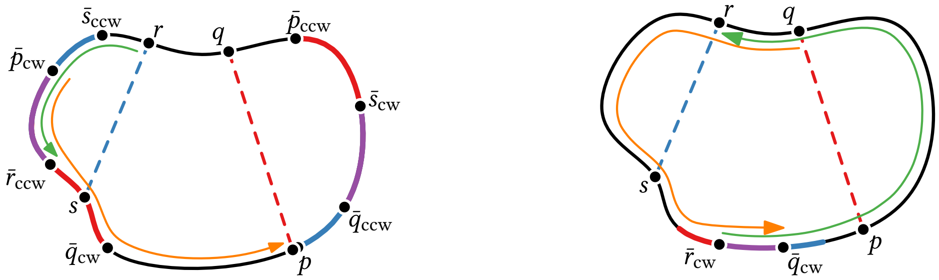
$$
\begin{aligned}
d_{\text{ccw}}(q, p) + |pq| + d_{\text{ccw}}(r, s) + |rs| &= 2d_{\text{ccw}}(\bar{p}_{\text{cw}}, p) + 2d_{\text{ccw}}(r, \bar{r}_{\text{ccw}}) \geq 2d_{\text{ccw}}(r, p) \\
\iff \quad |pq| + |rs| &\geq \underbrace{d_{\text{ccw}}(r, p) - d_{\text{ccw}}(q, p)}_{=d_{\text{ccw}}(r, q)} + \underbrace{d_{\text{ccw}}(r, p) - d_{\text{ccw}}(r, s)}_{=d_{\text{ccw}}(s, p)} \quad .
\end{aligned}
$$

Suppose the unaffected regions of pq and rs overlap along the hourglass. Analogously to the overlap on the bowtie, we derive that this occurs precisely when $|pq|+|rs| \geq d_{\mathrm{ccw}}(p,r)+d_{\mathrm{ccw}}(q,s)$.

If $|pq| = d(p,q)$ or $|rs| = d(r,s)$, then the shortcuts pq and rs cannot be useful for the cycle C. Therefore, the shortcuts pq and rs are useful for the cycle C if and only if $|pq| < d(p,q)$ and $|rs| < d(r,s)$ and $|pq| + |rs| < d_{\mathrm{ccw}}(r,q) + d_{\mathrm{ccw}}(s,p)$ and $|pq| + |rs| < d_{\mathrm{ccw}}(p,r) + d_{\mathrm{ccw}}(q,s)$. □

Lemma 2.4. *Two consecutive shortcuts pq and rs are useful for a cycle C if and only if $|pq| < d(p,q)$ and $|rs| < d(r,s)$ and $|pq| + |rs| < |d_{ccw}(s,p) - d_{ccw}(q,r)|$.*

Proof. Suppose pq and rs are useless consecutive shortcuts with $|pq| < d(p,q)$ and $|rs| < d(r,s)$. We assume $d_{\mathrm{ccw}}(q,r) \leq d_{\mathrm{ccw}}(s,p)$, without loss of generality. This means the unaffected regions of pq and rs overlap along the handset or along the base station, as illustrated in Figure 2.10.



(a) An overlap on the handset.

(b) An impossible overlap on the base station.

Figure 2.10: The two cases of useless consecutive shortcuts with overlapping unaffected regions.

Suppose the unaffected regions of pq and rs overlap on the handset. This overlap consists of the clockwise paths from $\bar{r}_{\mathrm{ccw}}$ to $\bar{p}_{\mathrm{cw}}$ and, symmetrically, from $\bar{s}_{\mathrm{cw}}$ to $\bar{q}_{\mathrm{ccw}}$. An overlap along a clockwise path from $\bar{p}_{\mathrm{ccw}}$ to $\bar{s}_{\mathrm{ccw}}$ is impossible, as $\bar{p}_{\mathrm{ccw}}$ would lie on $C_{\mathrm{cw}}(p,q)$ or $\bar{s}_{\mathrm{ccw}}$ on $C_{\mathrm{cw}}(r,s)$. The sum of the lengths of the counter-clockwise paths from r to $\bar{r}_{\mathrm{ccw}}$ and from $\bar{p}_{\mathrm{cw}}$ to p is at least the length of the counter-clockwise path from r to p, i.e., $d_{\mathrm{ccw}}(\bar{p}_{\mathrm{cw}},p) + d_{\mathrm{ccw}}(r,\bar{r}_{\mathrm{ccw}}) \geq d_{\mathrm{ccw}}(r,p)$. This is equivalent to $|pq| + |rs| \geq d_{\mathrm{ccw}}(s,p) - d_{\mathrm{ccw}}(q,r)$, since

$$\begin{aligned} d_{\mathrm{ccw}}(q,p) + |pq| + d_{\mathrm{ccw}}(r,s) + |rs| &= 2d_{\mathrm{ccw}}(\bar{p}_{\mathrm{cw}},p) + 2d_{\mathrm{ccw}}(r,\bar{r}_{\mathrm{ccw}}) \geq 2d_{\mathrm{ccw}}(r,p) \\ \iff |pq| + |rs| &\geq \underbrace{d_{\mathrm{ccw}}(r,p) - d_{\mathrm{ccw}}(r,s)}_{=d_{\mathrm{ccw}}(s,p)} + \underbrace{d_{\mathrm{ccw}}(r,p) - d_{\mathrm{ccw}}(q,p)}_{=-d_{\mathrm{ccw}}(q,r)} . \end{aligned}$$

Suppose the unaffected regions of pq and rs overlap on the base station. This overlap consists of a clockwise path from $\bar{q}_{\mathrm{cw}}$ to $\bar{r}_{\mathrm{cw}}$, as illustrated in Figure 2.10b. The sum of the lengths of the clockwise paths from r to $\bar{r}_{\mathrm{cw}}$ and from $\bar{q}_{\mathrm{cw}}$ to q is at least the length of the cycle C, i.e., $d_{\mathrm{ccw}}(\bar{r}_{\mathrm{cw}},r) + d_{\mathrm{ccw}}(q,\bar{q}_{\mathrm{cw}}) \geq |C| + d_{\mathrm{ccw}}(q,r) \geq |C|$. This yields $|pq| + |rs| \geq d(p,q) + d(r,s)$, since

$$\begin{aligned} d_{\mathrm{ccw}}(q,p) + |pq| + d_{\mathrm{ccw}}(s,r) + |rs| &= 2d_{\mathrm{ccw}}(q,\bar{q}_{\mathrm{cw}}) + 2d_{\mathrm{ccw}}(\bar{r}_{\mathrm{cw}},r) \geq 2|C| \\ \implies |pq| + |rs| &\geq \underbrace{|C| - d_{\mathrm{ccw}}(q,p)}_{=d_{\mathrm{ccw}}(p,q) \geq d(p,q)} + \underbrace{|C| - d_{\mathrm{ccw}}(s,r)}_{=d_{\mathrm{ccw}}(r,s) \geq d(r,s)} . \end{aligned}$$

The inequality $|pq| + |rs| \geq d(p,q) + d(r,s)$ contradicts $|pq| < d(p,q)$ and $|rs| < d(r,s)$. Therefore, the unaffected regions in the consecutive case can only overlap along the handset. Thus, pq and rs are useful if and only if $|pq| < d(p,q)$, $|rs| < d(r,s)$, and $|pq| + |rs| < |d_{\text{ccw}}(s,p) - d_{\text{ccw}}(q,r)|$. □

Theorem 2.5. *For every pair pq and rs of consecutive shortcuts for a geometric cycle C, there exists a pair $p'q'$ and $r's'$ of alternating shortcuts with* $\operatorname{diam}(C + p'q' + r's') \leq \operatorname{diam}(C + pq + rs)$.

Proof. Suppose pq and rs are useful consecutive shortcuts. We assume, without loss of generality, $d_{\text{ccw}}(q,r) \leq d_{\text{ccw}}(s,p)$ and $d_{\text{ccw}}(p,q) \leq d_{\text{ccw}}(r,s)$. Otherwise, we swap p, q, r, and s, appropriately.

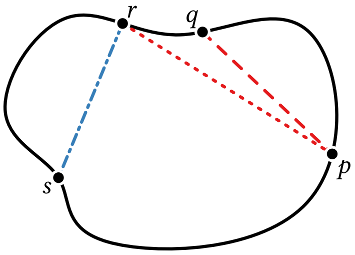


Figure 2.11: Replacing consecutive shortcuts pq and rs with $p'q' = pr$ and $r's' = rs$.

The touching shortcuts pr and rs are both alternating and consecutive, as illustrated in Figure 2.11. We first argue that pr and rs are useful shortcuts for the cycle C and then we match each candidate diametral cycle in $C + pq + rs$ with one candidate diametral cycle in $C + pr + rs$ of smaller or equal size.

Since pq and rs are useful consecutive shortcuts with $d_{\text{ccw}}(q,r) \leq d_{\text{ccw}}(s,p)$, we have $|pq| < d(p,q)$ and $|rs| < d(r,s)$ and $|pq| + |rs| + d_{\text{ccw}}(q,r) < d_{\text{ccw}}(s,p)$, by Lemma 2.4.

According to Lemma 2.3, the alternating pair of shortcuts pr and rs is useful for the cycle C, because we have $|pr| < d(p,r)$, due to $|pq| < d(p,q)$, and $|rs| < d(r,s)$, as well as

$$|pr| + |rs| < d(p,r) + d(r,s) \leq d_{\text{ccw}}(p,r) + d_{\text{ccw}}(r,s) \ ,$$
$$\text{and } |pr| + |rs| \leq |pq| + d(q,r) + |rs| < d_{\text{ccw}}(s,p) \leq d_{\text{ccw}}(s,p) + d_{\text{ccw}}(r,r) \ .$$

We match the candidates for the diametral cycles in $C + pq + rs$ with those in $C + pr + rs$. The bowtie in $C + pr + rs$ is at most as long as the handset in $C + pq + rs$, the hourglass in $C + pr + rs$ is at most as long as the base station in $C + pq + rs$, the blue split cycle did not change, and the red split cycle in $C + pr + rs$ is at most as long as the red split cycle in $C + pq + rs$.

We show the claim about the red split cycle. The red split cycle remains on the same side of the red shortcut when replacing pq with pr, due to the following. Our premises $d_{\text{ccw}}(p,q) \leq d_{\text{ccw}}(r,s)$ and $d_{\text{ccw}}(q,r) \leq d_{\text{ccw}}(s,p)$ imply $|C_{\text{ccw}}(p,q)| \leq |C_{\text{cw}}(p,q)|$, since $d_{\text{ccw}}(p,q) \leq d_{\text{ccw}}(r,s) \leq d_{\text{ccw}}(q,p)$, and $|C_{\text{ccw}}(p,r)| \leq |C_{\text{cw}}(p,r)|$, since $d_{\text{ccw}}(p,r) = d_{\text{ccw}}(p,q) + d_{\text{ccw}}(q,r) \leq d_{\text{ccw}}(r,s) + d_{\text{ccw}}(s,p) = d_{\text{ccw}}(r,p)$. Moreover, the red split cycle shrinks when moving q to r, due to the triangle inequality and the fact that the red split cycle remains on the same side.

The continuous diameter of $C + pq + rs$ is half of the maximum of the lengths of the four candidates for diametral cycles (excluding C, since pq and rs are useful). When replacing pq with pr, we decrease or maintain each of these four values whose maximum determines $\operatorname{diam}(C{+}pr{+}rs)$, since pr and rs are useful shortcuts. Therefore, $\operatorname{diam}(C + pr + rs) \leq \operatorname{diam}(C + pq + rs)$ □

2.3 Balancing Diametral Cycles

We show that every cycle has an optimal pair of alternating shortcuts where the bowtie and the hourglass are both diametral and we show that every convex cycle has an optimal pair of shortcuts where both split cycles are diametral as well. We obtain these results by applying a sequence of operations to some initial pair of alternating shortcuts. Each operation brings the candidate diametral cycles closer to the desired balance by sliding the endpoints of the shortcuts along the cycle in a way that reduces or maintains the continuous diameter. The last two operations only reduce the diameter for convex cycles as the shortcuts might get stuck at reflex vertices, which leads to our characterization of optimal shortcuts for convex and non-convex cycles.

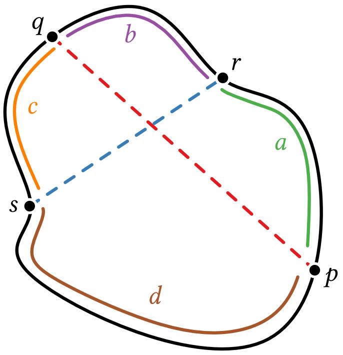


Figure 2.12: The four sections of a cycle for a pair of alternating shortcuts.

Let pq and rs be two alternating shortcuts and let $a = d_{\mathrm{ccw}}(p, r)$, $b = d_{\mathrm{ccw}}(r, q)$, $c = d_{\mathrm{ccw}}(q, s)$, and $d = d_{\mathrm{ccw}}(s, p)$. As depicted in Figure 2.12, we assume that the red split cycle contains s and the blue split cycle contains p, i.e., $a + b \le c + d$ and $b + c \le a + d$. We denote the lengths of the bowtie ($\bowtie$), the hourglass ($⧖$), the red split cycle ($⦸$), and the blue split cycle ($\oslash$) as follows.

$$\bowtie := a + c + |pq| + |rs| \qquad ⦸ := c + d + |pq| \qquad ⧖ := b + d + |pq| + |rs| \qquad \oslash := a + d + |rs|$$

Lemma 2.6. *For each relation $\sim \in \{<, =, >\}$, we have*

$$\begin{aligned} \bowtie \sim ⧖ &\iff a + c \sim b + d & ⦸ \sim \oslash &\iff c + |pq| \sim a + |rs| \\ \bowtie \sim ⦸ &\iff a + |rs| \sim d & ⧖ \sim ⦸ &\iff b + |rs| \sim c \\ \bowtie \sim \oslash &\iff c + |pq| \sim d & ⧖ \sim \oslash &\iff b + |pq| \sim a \end{aligned}$$

and the shortcuts pq and rs are useful for the geometric cycle C if and only if we have

$$|pq| < a + b \text{ and } |rs| < b + c \text{ and } |pq| + |rs| < a + c \text{ and } |pq| + |rs| < b + d \ .$$

Proof. The relations between $\bowtie$, $⧖$, $⦸$, and $\oslash$ follow from their definitions. For instance,

$$\bowtie \sim ⧖ \iff a + c + |pq| + |rs| \sim b + d + |pq| + |rs| \iff a + c \sim b + d \ .$$

The conditions for pq and rs being useful rephrase Lemma 2.3 in terms of the cycle sections. □

We prove our main structural result: For every geometric cycle C, there exists an optimal pair of shortcuts in alternating configuration where the bowtie and the hourglass are both diametral and where each split cycle is either diametral as well, or has one endpoint at a reflex vertex. For convex cycles, all four candidate cycles are diametral for every optimal pair of shortcuts.

Our approach is as follows. We start with some optimal pair of alternating shortcuts and move them in a way that brings us closer to the desired properties while maintaining the shortcuts in alternating configuration and without increasing the continuous diameter in the process.

Formally, a *movement* of pair of shortcuts pq and rs for a cycle C to another pair of shortcuts $p'q'$ and $r's'$ for C is a continuous function $\gamma : [0,1] \to C^4$ with $\gamma(0) = (p,q,r,s)$ and $\gamma(1) = (p',q',r',s')$. A movement of a pair of shortcuts $\gamma(t) = (p(t), q(t), r(t), s(t))$ with $t \in [0,1]$ is *save* when $p(t)q(t)$ and $r(t)s(t)$ form an alternating pair of useful shortcuts for all $t \in [0,1]$, and when the continuous diameter is weakly decreasing throughout the movement, i.e., for all $t, t' \in [0,1]$ with $t \le t'$, we have $\operatorname{diam}(C + p(t)q(t) + r(t)s(t)) \ge \operatorname{diam}(C + p(t')q(t') + r(t')s(t'))$.

Suppose pq and rs are a useful pair of alternating shortcuts for a geometric cycle C. The following results are the stepping stones towards our main structural result.

1. If the augmented cycle $C + pq + rs$ has exactly one diametral cycle, then we can move the shortcuts safely until there are at least two diametral cycles.
2. If the bowtie and hourglass in $C + pq + rs$ have different lengths, then we can move the shortcuts safely until the bowtie and the hourglass have the same length.
3. There exists an optimal pair of augmented shortcuts pq and rs for C such that $C + pq + rs$ has at least two diametral cycles and the bowtie and the hourglass have equal length.
4. If both split cycles are diametral in $C + pq + rs$ and the hourglass and bowtie are in balance, then we can safely move the shortcuts until all four candidate cycles are diametral.
5. There exists an optimal pair of alternating shortcuts pq and rs for C such that the bowtie and the hourglass are diametral in the augmented cycle $C + pq + rs$.
6. If the bowtie and the hourglass are diametral in $C + pq + rs$, then we can move the shortcuts safely until each split cycle is either diametral as well, or at least one endpoint of its corresponding shortcut is stuck at some reflex vertex of C.

To establish the first claim, we show that a split cycle is never the only diametral cycle when its shortcut splits the cycle into two paths of equal length. We say a split cycle divides the cycle *evenly* when the corresponding shortcut xy divides C evenly, i.e., when the clockwise path from x to y has the same length as the counter-clockwise path from x to y, i.e., $d_{\text{ccw}}(x,y) = d_{\text{cw}}(x,y)$.

Lemma 2.7. *If the red split cycle or the blue split cycle of a pair of useful alternating shortcuts evenly divides the cycle, then this split cycle must have a length of at most* $\bowtie$ *or at most* ⧗.

Proof. We assume, for the sake of a contradiction, that there exists a useful alternating pair of shortcuts pq and rs, where the red split cycle divides the cycle evenly, i.e., $a + b = c + d$, and where the bowtie and the hourglass are shorter than the red split cycle, i.e., $\bowtie <$ ⦸ and ⧗ $<$ ⦸. Lemma 2.6 implies $a + |rs| < d$ and $b + |rs| < c$, which yields the contradiction $|rs| < 0$, since

$$a + |rs| < d = a + b - c < a - |rs| \implies 2|rs| < 0 \ .$$

Therefore, an even red or blue split cycle is always shorter than the bowtie or the hourglass. □

Lemma 2.8. *For every geometric cycle C, there exists an optimal pair pq and rs of shortcuts in alternating configuration such that neither split cycle is the only diametral cycle in $C + pq + rs$.*

Proof. Suppose pq and rs are useful alternating shortcuts where the blue split is the only diametral cycle in $C + pq + rs$, i.e., $\mathrm{diam}(C + pq + rs) = \oslash/2$ with $\bowtie < \oslash$, $\text{⧗} < \oslash$, and $\obslash < \oslash$.

The blue split cycle cannot divide the cycle evenly, due to Lemma 2.7, since it is diametral. This means we can shrink the blue split cycle by moving r clockwise towards p until we arrive at some position r' where another diametral cycle appears. We denote the bowtie, the hourglass, the red split cycle, and the blue split cycle in $C + pq + r's$ by $\bowtie'$, $\text{⧗}'$, $\obslash'$, and $\oslash'$, respectively. Moving r to r' leads to the following changes for the candidates for the diametral cycles.

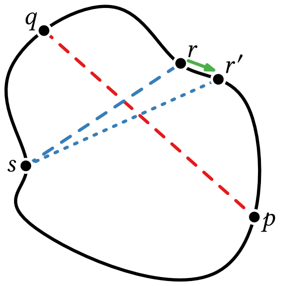


Figure 2.13: Shrinking the blue split cycle by moving r clockwise to r'.

- The length of the blue split cycle decreases or remains unchanged, i.e., $\oslash \geq \oslash'$.
- The length of the red split cycle remains unchanged, i.e., $\obslash = \obslash'$.
- The bowtie and the blue split cycle change proportionally, i.e., $\bowtie \geq \bowtie'$ and $\bowtie' < \oslash'$.
- The length of the hourglass remains unchanged or increases, i.e., $\text{⧗} \leq \text{⧗}'$, and it increases when the length of the blue split cycle does not change, i.e, $\oslash - \text{⧗} > \oslash' - \text{⧗}'$.

This means $\oslash' = \text{⧗}'$ or $\oslash' = \obslash'$ and the continuous diameter decreases or remains the same, i.e., $\mathrm{diam}(C + pq + r's) \leq \mathrm{diam}(C + pq + rs)$, provided that pq and $r's$ are useful and alternating.

We argue that r' lies on the clockwise path from r to p, which implies that pq and $r's$ are alternating shortcuts. With Lemma 2.6, $\text{⧗} < \oslash$ implies $d_{\mathrm{ccw}}(r, q) + |pq| < d_{\mathrm{ccw}}(p, r)$ and $\obslash < \oslash$ implies $d_{\mathrm{ccw}}(q, s) + |pq| - |rs| < d_{\mathrm{ccw}}(p, r)$. Since $d_{\mathrm{ccw}}(q, p) + |pq| > 0$, there is some position r'' along the clockwise path from r to p with $d_{\mathrm{ccw}}(r'', q) + |pq| = d_{\mathrm{ccw}}(p, r'')$, by the intermediate value theorem. Therefore, r' lies on the clockwise path from r to p, because r' is the first position clockwise from r where $d_{\mathrm{ccw}}(r', q) + |pq| = d_{\mathrm{ccw}}(p, r')$ or $d_{\mathrm{ccw}}(q, s) + |pq| - |r's| = d_{\mathrm{ccw}}(p, r')$.

We argue that pq and $r's$ are useful. Since pq and rs are useful, we have

$$|pq| < a + b \text{ and } |rs| < b + c \text{ and } |pq| + |rs| < a + c \text{ and } |pq| + |rs| < b + d \ ,$$

where $a = d_{\mathrm{ccw}}(p, r)$, $b = d_{\mathrm{ccw}}(r, q)$, $c = d_{\mathrm{ccw}}(q, s)$, and $d = d_{\mathrm{ccw}}(s, r)$, as illustrated in Figure 2.12. Let a', b', c', and d' be the lengths of the sections into which p, r', q, and s subdivide C, i.e.,

$$\begin{aligned} a' &= d_{\mathrm{ccw}}(p, r') = a - d_{\mathrm{ccw}}(r', r) & c' &= d_{\mathrm{ccw}}(q, s) = c \\ b' &= d_{\mathrm{ccw}}(r', q) = b + d_{\mathrm{ccw}}(r', r) & d' &= d_{\mathrm{ccw}}(s, p) = d \ . \end{aligned}$$

We have $a' + b' \leq c' + d'$ and $b' + c' \leq a' + d'$, i.e., $d(p, q) = a' + b'$ and $d(r, s) = b' + c'$, because

$$\begin{aligned} a' + b' &= a - d_{\mathrm{ccw}}(r', r) + b + d_{\mathrm{ccw}}(r', r) = a + b \leq c + d = c' + d' \ , \\ b' + c' &= b + d_{\mathrm{ccw}}(r', r) + c \leq a + d + d_{\mathrm{ccw}}(r', r) = a' + d' \ . \end{aligned}$$

This means that the shortcuts pq and $r's$ are arranged as depicted in Figure 2.12, i.e., the red split cycle of $C + pq + r's$ contains s and the blue split cycle of $C + pq + r's$ contains p. Therefore, Lemma 2.6 applies and, thus, the shortcuts pq and $r's$ are useful if and only if

$$|pq| < a' + b' \text{ and } |r's| < b' + c' \text{ and } |pq| + |r's| < a' + c' \text{ and } |pq| + |r's| < b' + d' \ .$$

We have $|pq| < a' + b'$, since $|pq| < a + b = d(p, q) = a' + b'$ and we have $|r's| < b' + c'$, since

$$|r's| \leq |r'r| + |rs| \leq d_{\text{ccw}}(r, r') + |rs| < d_{\text{ccw}}(r, r') + b + c = b' + c = b' + c' \ ,$$

we have $|pq| + |r's| < b' + d'$, since

$$|pq| + |r's| \leq |pq| + |rs| + d_{\text{ccw}}(r', r) < b + d + d_{\text{ccw}}(r', r) = b' + d = b' + d' \ ,$$

and $|pq| + |r's| < a' + c'$, as $⧖' \leq \oslash'$ implies $b' + |pq| \leq a'$ and $|pq| + |r's| < |pq| + b' + c' \leq a' + c'$.

Therefore, pq and $r's$ are useful alternating shortcuts such that the blue split is not the only diametral path in $C + pq + r's$. Moreover, if pq and rs was an optimal pair of shortcuts for C, then so are pq and $r's$, since $\operatorname{diam}(C + pq + r's) \leq \operatorname{diam}(C + pq + rs)$. Therefore, there exists an optimal pair of shortcuts where the blue split cycle is not the only diametral cycle. Analogously, there exists a pair of optimal shortcuts where the red split cycle is not the only diametral cycle. □

Lemma 2.9. *For every cycle, there exists a pair of optimal alternating shortcuts with* $\bowtie = ⧖$.

Proof. Suppose pq and rs are useful alternating shortcuts with $\bowtie \neq ⧖$. We balance $\bowtie$ and $⧖$ using the following operations that maintain or decrease the continuous diameter while decreasing the difference between bowtie and hourglass. Two of these operations are illustrated in Figure 2.14.

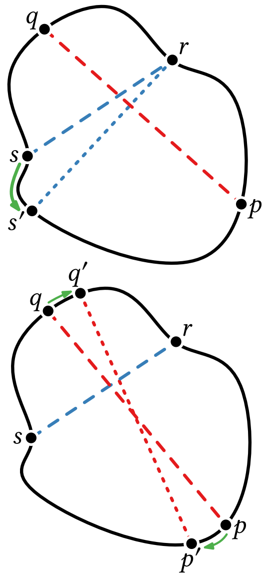


Figure 2.14: Ops 1.a and 2.a.

1. As long as neither the red nor the blue split cycle divides the cycle C evenly, we shrink the larger split cycle in a way that decreases the difference of bowtie and hourglass:
    a) If $\bowtie < ⧖$ and $\obslash \leq \oslash$, move s counter-clockwise.
    b) If $\bowtie < ⧖$ and $\obslash \geq \oslash$, move p clockwise.
    c) If $\bowtie > ⧖$ and $\obslash \leq \oslash$, move r clockwise.
    d) If $\bowtie > ⧖$ and $\obslash \geq \oslash$, move q counter-clockwise.
2. Once a split cycle evenly divides the cycle, we move the endpoints of the corresponding shortcut in the direction that decreases the difference between bowtie and hourglass:
    a) If $\bowtie < ⧖$ and $\obslash$ is even, move p and q clockwise.
    b) If $\bowtie > ⧖$ and $\obslash$ is even, move p and q counter-clockwise.
    c) If $\bowtie < ⧖$ and $\oslash$ is even, move s and r counter-clockwise.
    d) If $\bowtie > ⧖$ and $\oslash$ is even, move s and r clockwise.

For each operation, we argue that pq and rs remain useful alternating shortcuts and that the diameter never increases while the difference between hourglass and bowtie always decreases.

The overlap between the cases for Operation 1 is intentional: if $⦸ = \oslash$, then we move the endpoints of the shortcuts with appropriate speeds to maintain this balance. For Operation 2, there is nothing to do when both split cycles divide the cycle evenly, since then $\bowtie = ⧗$.

Suppose $\bowtie < ⧗$ and neither split is even. We apply Operations 1.a and 1.b, i.e., we move s counterclockwise to s' and we move p clockwise to p' until bowtie and hourglass are in balance or until one split is even. Since we never move past an even split, we have $d(p', q) = d_{\text{ccw}}(p', q) \leq d_{\text{cw}}(p', q)$ and $d(r, s') = d_{\text{ccw}}(r, s') \leq d_{\text{cw}}(r, s')$. This means $p'q$ and rs' are alternating shortcuts where the red split cycle contains s' and the blue split cycle contains p'. We denote the bowtie, the hourglass, the red split cycle, and the blue split cycle in $C + p'q + r's$ by $\bowtie'$, $⧗'$, $⦸'$, and $\oslash'$, respectively. The candidate diametral cycles change as follows.

- The red and blue split cycles shrink or remain the same, i.e., $⦸ \geq ⦸'$ and $\oslash \geq \oslash'$.
- The bowtie grows or remains the same, i.e., $\bowtie \leq \bowtie'$.
- The hourglass remains the same or shrinks, i.e., $⧗ \geq ⧗'$.
- The bowtie grows when the hourglass remains unchanged and, conversely, the hourglass shrinks when the bowtie remains unchanged, i.e., $⧗ - \bowtie > ⧗' - \bowtie'$.

Therefore, the continuous diameter decreases or remains the same, i.e., $\text{diam}(C + p'q + rs') \leq \text{diam}(C + pq + rs)$, provided that $p'q$ and rs' remain useful shortcuts.

We argue that $p'q$ and $r's$ are useful. Since pq and rs are useful, we have

$$|pq| < a + b \text{ and } |rs| < b + c \text{ and } |pq| + |rs| < a + c \text{ and } |pq| + |rs| < b + d \ ,$$

where $a = d_{\text{ccw}}(p, r)$, $b = d_{\text{ccw}}(r, q)$, $c = d_{\text{ccw}}(q, s)$, and $d = d_{\text{ccw}}(s, r)$, as illustrated in Figure 2.12. Let a', b', c', and d' be the lengths of the sections into which p', r, q, and s' subdivide C, i.e.,

$$\begin{aligned} a' &= d_{\text{ccw}}(p', r) = a + d_{\text{ccw}}(p', p) & c' &= d_{\text{ccw}}(q, s') = c + d_{\text{ccw}}(s, s') \\ b' &= d_{\text{ccw}}(r, q) = b & d' &= d_{\text{ccw}}(s', p') = d - d_{\text{ccw}}(p', p) - d_{\text{ccw}}(s, s') \ . \end{aligned}$$

We have $a' + b' \leq c' + d'$ and $b' + c' \leq a' + d'$, i.e., $d(p', q) = a' + b'$ and $d(r, s') = b' + c'$, as argued above. Therefore, Lemma 2.6 applies. We have $|p'q| < a' + b'$ and $|rs'| < b' + c'$, since

$$\begin{aligned} |p'q| &\leq d(p', p) + |pq| < d(p', p) + a + b = a' + b = a' + b' \ , \\ |rs'| &\leq d(s, s') + |rs| < d(s, s') + b + c = b + c' = b' + c' \ . \end{aligned}$$

We have $|p'q| + |rs'| < \min\{a' + c', b' + d'\}$, since $\bowtie' \leq ⧗'$ implies $a' + c' \leq b' + d'$ and

$$|p'q| + |rs'| \leq |pq| + d(p', p) + |rs| + d(s, s') < a + d(p', p) + c + d(s, s') = a' + c' \ .$$

Therefore, $p'q$ and rs' form a useful pair of alternating shortcuts, due to Lemma 2.6. This means that applying Operations 1.a and 1.b never increases the diameter while reducing the difference between bowtie and hourglass. Similarly, the same holds for Operations 1.c and 1.d.

Suppose $\bowtie < ⧗$ and the red split cycle is an even split. We apply Operation 2.a, i.e., we move p and q at the same speed clockwise until (1) the bowtie and the hourglass are in balance or (2) the shortcuts cease to be alternating or (3) the role of the candidate diameteral cycles changes. We argue that the first condition holds when this rotation of the red shortcut ends. Let p' and q' be the positions of p and q, respectively, where the rotation ends. We denote the bowtie, the hourglass, the red split cycle, and the blue split cycle in $C + p'q' + rs$ by $\bowtie'$, $⧖'$, $\obslash'$, and $\oslash'$, respectively. The candidate diametral cycles change as follows.

- The red split cycle remains an even split; it shrinks and grows with the red shortcut. The red split cycle cannot determine the diameter, since $\obslash' \leq \bowtie'$ or $\obslash' \leq ⧖'$, due to Lemma 2.7.
- The blue split cycle remains unchanged, i.e., $\oslash = \oslash'$.
- The bowtie grows or remains the same, i.e., $\bowtie \leq \bowtie'$.
- The hourglass remains the same or shrinks, i.e., $⧖ \geq ⧖'$.
- The bowtie grows when the hourglass remains unchanged and, conversely, the hourglass shrinks when the bowtie remains unchanged, i.e., $⧖ - \bowtie > ⧖' - \bowtie'$.

This means $\operatorname{diam}(C + p'q' + rs) \leq \operatorname{diam}(C + pq + rs)$, provided that $p'q'$ and rs are useful. We argue that this is the case after establishing that the clockwise turn of the red shortcut ends with the hourglass and the bowtie being in balance. Let $\delta = d_{\mathrm{ccw}}(q', q) = d_{\mathrm{ccw}}(p', p)$ and let a', b', c', and d' be the lengths of the sections into which p', r, q', and s subdivide C, i.e.,

$$\begin{aligned} a' &= d_{\mathrm{ccw}}(p', r) = a + \delta & \qquad c' &= d_{\mathrm{ccw}}(q', s) = c + \delta \\ b' &= d_{\mathrm{ccw}}(r, q') = b - \delta & \qquad d' &= d_{\mathrm{ccw}}(s, p') = d - \delta \quad . \end{aligned}$$

We argue that the roles of the candidates for the diametral cycles remain the same throughout the rotation of the red shortcut by showing $a' + b' \leq b' + c'$ and $b' + c' < a' + d'$. First, we have $a' + b' = b' + c'$, since the red split cycle remains an equal split. Second, the blue split cycle in $C + pq + rs$ cannot split the cycle evenly, since $\bowtie < ⧖$. Therefore, we have $b + c < a + d$ and

$$b' + c' = (b - \delta) + (c + \delta) = b + c < a + d = (a + \delta) + (d - \delta) = a' + d' \quad .$$

Therefore, s lies on the red split cycle in $C + p'q' + rs$ and p' lies on the blue split cycle in $C + p'q' + rs$ and, thus, the roles of the candidates for diametral cycles remain the same.

We argue that $p'q'$ and rs form a pair of alternating shortcuts. The observations $a + b = c + d$ and $b + c < a + d$ imply $b < d$. If the rotation of the red shortcut ends with the shortcuts ceasing to be alternating, then $q' = r$, i.e., $\delta = b$. Since pq and rs are useful, we have $|rs| < d(r, s) = c + b$ and, thus, $b + c > 0$. Therefore, Lemma 2.6 implies $\bowtie' > ⧖'$, since $a' + c' > b' + d'$, because

$$a' + c' = (a + b) + (c + b) > a + b = c + d \geq 0 + d \geq (b - b) + (d - b) = b' + d' \quad .$$

By the intermediate value theorem, there was some position of the red shortcut where the hourglass and the bowtie were balanced before q' could reach r. Therefore, the rotation of the red shortcut ends with q' along the interior of the clockwise path from q to r and p' along the interior of the clockwise path from p to s. Hence, $p'q'$ and rs are alternating shortcuts.

Therefore, the rotation of the red shortcut ends when the bowtie and hourglass are in balance. It remains to show that $p'q'$ and rs are useful. Since pq and rs are useful, we have

$$|pq| < a + b \text{ and } |rs| < b + c \text{ and } |pq| + |rs| < a + c \text{ and } |pq| + |rs| < b + d \enspace .$$

By the triangle inequality, we have $|p'q'| \leq d(p', q') = a' + b'$. We have $|p'q'| < a' + b'$, since otherwise the cycle C could not possess a useful pair of shortcuts, because it would contain a line segment of length $|C|/2$, since $a' + b' = c' + d' = |C|/2$. However, pq and rs form a useful pair of shortcuts for C. We have $|p'q'| + |rs| < a' + c'$ and $|p'q'| + |rs| < b' + d'$, because

$$|p'q'| + |rs| \leq |pq| + 2\delta + |rs| < (a + \delta) + (c + \delta) = a' + c' = b' + d' \enspace .$$

According to Lemma 2.6, this means that $p'q'$ and rs form a useful pair of shortcuts for C.

Operation 2.a never increases the diameter while reducing the difference between bowtie and hourglass until they are equal. Similarly, we can argue that the same holds for Operations 2.b, 2.c, and 2.e. The above implies the claim, since, by using Operation 1 and Operation 2, we can transform every alternating configuration of useful shortcuts into another configuration where bowtie and hourglass have equal length without increasing the continuous diameter. □

Corollary 2.10. *For every cycle, there exists an optimal pair of alternating shortcuts where none of the split cycles is the only diametral cycle and the bowtie and the hourglass have the same length.*

Proof. Lemma 2.8 yields an optimal pair of alternating shortcuts pq and rs for a cycle C where none of the split cycles is the only diametral cycle. We begin with this pair of shortcuts.

If the bowtie and hourglass are unbalanced in $C + pq + rs$, then we apply the operations from Lemma 2.9. These operations maintain the invariant that if one of the split cycles is diametral, then some other candidate cycle is diametral as well: If neither split cycle is even, then Operations 1.a to 1.d shrink the largest split cycles at the same rate as they shrink the larger of bowtie and hourglass. If one of the split cycles is even, then it cannot become diametral by Lemma 2.7. Operations 2.a to 2.d rotate the split cycle that is even and leave the other one unchanged. If both split cycles are even, then the bowtie and hourglass are already in balance.

Therefore, we obtain an optimal pair of alternating shortcuts where none of the split cycles is the only diametral cycle and the bowtie and the hourglass have the same length. □

Theorem 2.11. *For every geometric cycle, there exists an optimal pair of shortcuts in alternating configuration such that the hourglass and the bowtie are both diametral, i.e.,* $\bowtie = ⧖ \geq ⦸, \oslash$.

Proof. We consider a geometric cycle C. If C is a degenerate cycle that does not posses any useful pair of shortcuts, then any pair of alternating shortcuts is optimal and the claim holds.

Suppose that C does possess some useful pair of shortcuts. Corollary 2.10 yields an optimal pair of alternating shortcuts pq and rs for C where neither split cycle is the only diametral cycle in $C + pq + rs$ and where the bowtie and the hourglass have equal length. If the bowtie and the hourglass are already diametral in $C + pq + rs$, then the claim follows, since $\bowtie = ⧖ \geq ⦸, \oslash$.

Suppose that neither the bowtie nor the hourglass are diametral cycles in $C + pq + rs$. Then, the cycle C cannot be diametral in $C + pq + rs$, because C possess some useful pair of shortcuts. This means that at least one of the split cycles is diametral in $C + pq + rs$ and, thus, $\obslash = \oslash > \bowtie = ⧖$, since neither split cycle can be the only diametral cycle. Since the bowtie and the hourglass are shorter than both split cycles, Lemma 2.7 implies that neither split cycle divides C evenly.

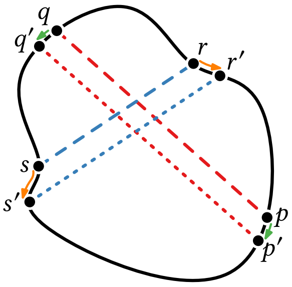


Figure 2.15: Shifting the shortcuts to shrink the split cycles while maintaining the balance between both split cycles and the balance between bowtie and hourglass.

We shrink both split cycles by simultaneously moving p and r clockwise while moving q and s counter-clockwise, as illustrated in Figure 2.15. We argue that we can shift pq and rs in a way that maintains the balance between the split cycles and the balance between bowtie and hourglass. If we shift p and q by some sufficiently small distance $\epsilon > 0$ to p' and q', i.e., $d_{\text{ccw}}(p', p) = d_{\text{ccw}}(q, q') = \epsilon$, then there exists some distance $\delta \geq 0$ such that the red split cycle and the blue split cycle have equal length if we shift r and s by distance δ to r' and s', i.e., $d_{\text{ccw}}(r', r) = d_{\text{ccw}}(s, s') = \delta$. This maintains the balance between bowtie and hourglass, since

$$a' + c' = (a + \delta - \epsilon) + (c + \epsilon - \delta) = a + c = b + d = (b + \delta + \epsilon) + (d - \delta - \epsilon) = b' + d' \ ,$$

where $a' = d_{\text{ccw}}(p', r')$, $b' = d_{\text{ccw}}(r', q')$, $c' = d_{\text{ccw}}(q', s')$, and $d' = d_{\text{ccw}}(s', p')$.

We stop shifting the shortcuts at the first positions $p'q'$ and $r's'$ where (1) all four candidate cycles are in balance or (2) two endpoints meet or (3) the role of the candidate cycles changes. Our goal is to argue that only the first event occurs and that $p'q'$ and $r's'$ are useful.

Let $a' = d_{\text{ccw}}(p', r')$, $b' = d_{\text{ccw}}(r', q')$, $c' = d_{\text{ccw}}(q', s')$, and $d' = d_{\text{ccw}}(s', p')$ be the lengths of the sections into which p', r', q', and s' divide the cycle C. We denote the bowtie, hourglass, red split cycle, and blue split cycle of the augmented cycle $C + p'q' + r's'$ by $\bowtie'$, $⧖'$, $\obslash'$, and $\oslash'$, respectively. Due to our invariants, we have $\obslash' = \oslash' \geq \bowtie' = ⧖'$ and, thus, $a' + c' = b' + d' = |C|/2$.

If one of the split cycles becomes an even split while we shift the shortcuts, then all four candidates for diametral cycles are in balance at that moment, due to Lemma 2.7. Therefore, we have $a' + b' \leq c' + d'$ and $b' + c' \leq a' + d'$ and, thus, $b' \leq d'$. This means that the roles of the candidates for diametral cycles cannot change, unless the endpoints of two cycles meet.

First, we argue that $q' \neq r'$ and $s' \neq p'$. Since pq and rs form a pair of useful alternating shortcuts for C, we have $0 < b$, since $0 \leq |pq| < a + b \leq b$. When we shift the shortcuts, we move r clockwise and q counter-clockwise. Therefore, we have $0 < b \leq b'$ and, thus, $q' \neq r'$. This also implies that $s' \neq p'$, since $0 < b' \leq d' < b' + d' = |C|/2$ implies $0 < d' = d(s', p')$.

Second, we argue that $r' \neq p'$ and $q' \neq s'$. The assumption $r' = p'$ leads to a contradiction, since then $a' = 0$ and, thus, $|C|/2 = a' + c' = c' < b' + c' \leq a' + d' = d' < b' + d' = |C|/2$. Therefore, we have $r' \neq q'$ and we obtain $q' \neq s'$ with a symmetric argument.

This means that the shift of the shortcuts ended with all four candidates for diametral cycles being in balance, i.e., $\bowtie' = ⧖' = ⦸' = \oslash'$. It remains to show that $p'q'$ and $r's'$ form a useful pair of shortcuts for C. We have $|p'q'| < d(p', q') = a' + b'$ and $|r's'| < d(r', s') = b' + c'$, since

$$|p'q'| \leq d(p', p) + |pq| + d(q, q') < d(p', p) + d(p, q) + d(q, q') = d(p', q')$$
$$|r's'| \leq d(r', r) + |rs| + d(s, s') < d(r', r) + d(r, s) + d(s, s') = d(r', s') \ .$$

Moreover, we have $|p'q'| + |r's'| < a' + c'$ and $|p'q'| + |r's'| < b' + d'$, since $\bowtie' = \oslash'$ implies $c' + |p'q'| = d'$ and $⧖' = ⦸'$ implies $b' + |r's'| = c'$ and, thus, with $b' > 0$ we obtain

$$|p'q'| + |r's'| = (d' - c') + (c' - b') = d' - b' < d' + b' = a' + c' \ .$$

This means that $p'q'$ and $r's'$ are useful alternating shortcuts with $\bowtie' = ⧖' = ⦸' = \oslash'$ such that $\operatorname{diam}(C + p'q' + r's') \leq \operatorname{diam}(C + pq + rs)$. Since pq and rs are optimal, so are $p'q'$ and $r's'$.

In summary, Corollary 2.10 yields an optimal pair alternating shortcuts where $\bowtie = ⧖ \geq ⦸, \oslash$ or $⦸ = \oslash > \bowtie = ⧖$. In the latter case, we shift the shortcuts to another optimal position where $\bowtie' = ⧖' = ⦸' = \oslash'$. Therefore, for every geometric cycle, there exists an optimal pair of alternating shortcuts where both the hourglass and the bowtie are diametral. □

Theorem 2.12. *For every convex cycle, there exists an optimal pair of alternating shortcuts such that the hourglass, the bowtie, and both split cycles are diametral, i.e.,* $\bowtie = ⧖ = ⦸ = \oslash$.

Proof. Let C be a convex cycle. According to Theorem 2.11, there exists an optimal pair of shortcuts pq and rs for C with $\bowtie = ⧖ \geq ⦸, \oslash$. Suppose we have $\bowtie = ⧖ > ⦸$ or $\bowtie = ⧖ > \oslash$.

We establish the claim in three steps. First, we show that we can grow each split cycle in a way that shortens its corresponding shortcut. Second, we grow the smaller split cycle until both split cycles have the same length. Third, we grow both split cycles simultaneously until their length matches the lengths of the bowtie and the hourglass. Each step maintains the balance between bowtie and hourglass, preserves usefulness, and never increases the continuous diameter.

We move p counter-clockwise and q by the same distance clockwise. As argued in the proof of Theorem 2.11, this shifts the red shortcut pq in a way that increases the size of the red split cycle, maintains the balance between bowtie and hourglass, and preserves usefulness.

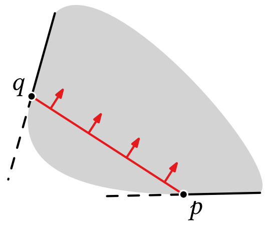


Figure 2.16: The case when shifting the red shortcut increases its length. Since the cycle is convex, shifting the red shortcut in the other direction must decrease its length.

Suppose that shifting the red shortcut pq to increase the red split cycle shortens the red shortcut. Since the cycle is convex, the red shortcut shrinks as we continue to shift it.

Suppose, on the other hand, that shifting the red shortcut pq to increase the red split cycle increases the length of the red shortcut, as illustrated in Figure 2.16. Since the cycle is convex, the red shortcut shrinks as we shift the red split cycle in the other direction, moving p clockwise and q counter-clockwise. Otherwise, p or q would be stuck at a reflex vertex. We shift the red shortcut to shrink the red split cycle until it becomes even and starts growing.

Since each shift operation maintains $a + c$ and $b + d$, the bowtie and the hourglass change as $|pq| + |rs|$ changes. As argued above, we shift pq or rs in a way that decreases $|pq| + |rs|$ and thereby $\bowtie$ and $⧖$ while increasing the smaller split until both splits are equal. The same applies when we continue to shift pq and rs at the same time adjusting the speed of the shifts to maintain $⦸ = \oslash$. Eventually, we arrive at shortcuts $p'q'$ and $r's'$ with $\bowtie' = ⧖' = ⦸' = \oslash'$ and $\mathrm{diam}(C + pq + rs) \geq \mathrm{diam}(C + p'q' + r's')$, since we only decreased the diametral cycles throughout the shift operations and maintain usefulness as argued in Theorem 2.11. □

Corollary 2.13. *For every geometric cycle, there exists an optimal pair of alternating shortcuts such that the hourglass and the bowtie are diametral and such that each split cycle is diametral or the shortcut of the split has at least one endpoint at a reflex vertex.* □

Lemma 2.14. *A pair of alternating shortcuts pq and rs is optimal for a non-degenerate geometric cycle if $|pq| + |rs|$ is minimal among all pairs of alternating shortcuts with $⧖ = \bowtie \geq ⦸, \oslash$.*

Proof. Suppose C is a non-degenerate geometric cycle. Then, there exists an optimal pair of useful alternating shortcuts p^*q^* and r^*s^* for C such that $⧖^* = \bowtie^* \geq ⦸^*, \oslash^*$. We have

$$\mathrm{diam}(C + p^*q^* + r^*s^*) = \frac{⧖^*}{2} = \frac{d(q^*, r^*) + d(s^*, p^*) + |p^*q^*| + |r^*s^*|}{2} = \frac{|C|}{4} + \frac{|p^*q^*| + |r^*s^*|}{2} \ ,$$

since $⧖^* = \bowtie^*$ implies $d(p^*, r^*) + d(q^*, s^*) = d(q^*, r^*) + d(s^*, p^*) = |C|/2$. Since p^*q^* and r^*s^* are useful for C, i.e., $\mathrm{diam}(C + p^*q^* + r^*s^*) < \mathrm{diam}(C) = |C|/2$, we have $|p^*q^*| + |r^*s^*| < |C|/2$.

Consider any pair of alternating shortcuts pq and rs that minimize the sum of the their lengths, i.e., $|pq| + |rs|$, such that $⧖ = \bowtie \geq ⦸, \oslash$. Note that we have $|pq| + |rs| \leq |p^*q^*| + |r^*s^*| < |C|/2$.

We argue that pq and rs are useful for C. Let $a = d_{\mathrm{ccw}}(p, r)$, $b = d_{\mathrm{ccw}}(r, q)$, $c = d_{\mathrm{ccw}}(q, s)$, and $d = d_{\mathrm{ccw}}(s, p)$ with $a + b \leq c + d$ and $b + c \leq a + d$. We need to show that $|pq| + |rs| < a + c$ and $|pq| + |rs| < b + d$ and $|pq| < a + b$ and $|rs| < b + c$. The first two conditions hold, since $\bowtie = ⧖$ implies $a + c = b + d = |C|/2$ and, thus, $|pq| + |rs| < |C|/2 = a + c = b + d$. The condition $|pq| < a + b$ holds, since $\bowtie \geq ⦸$ implies $a + |rs| \geq d$, i.e., $-|rs| \leq a - d$, and, thus,

$$|pq| < |C|/2 - |rs| \leq |C|/2 + a - d = a + b \ .$$

Likewise, the condition $|rs| < b + c$ holds, since $\bowtie \geq \oslash$ implies $-|pq| \leq c - d$ and, thus,

$$|rs| < |C|/2 - |pq| \leq |C|/2 + c - d = b + c \ .$$

This means pq and rs are useful for C and, thus, optimal, since

$$\mathrm{diam}(C + pq + rs) = \frac{|C|}{4} + \frac{|pq| + |rs|}{2} \leq \frac{|C|}{4} + \frac{|p^*q^*| + |r^*s^*|}{2} = \mathrm{diam}(C + p^*q^* + r^*s^*) \ .$$

Therefore, a pair of alternating shortcuts pq and rs is optimal for a non-degenerate geometric cycle if $|pq| + |rs|$ is minimal among all pairs of alternating shortcuts with $⧖ = \bowtie \geq ⦸, \oslash$. □

2.4 Algorithm

Guided by our structural results, we find an optimal pair of shortcuts for a geometric cycle C by searching for a shortest pair of alternating shortcuts pq and rs such that $\bowtie = ⧖ \geq \obslash, \oslash$. We divide this search into four parts depending on which shortcuts are stuck at a reflex vertex.

1. Neither shortcut is stuck at a reflex vertex and thus $\bowtie = ⧖ = \obslash = \oslash$.
2. Only the red shortcut pq is stuck at a reflex vertex and thus $\bowtie = ⧖ = \oslash > \obslash$.
3. Only the blue shortcut rs is stuck at a reflex vertex and thus $\bowtie = ⧖ = \obslash > \oslash$.
4. Both shortcuts are stuck at reflex vertices and thus $\bowtie = ⧖ > \obslash$ and $\bowtie = ⧖ > \oslash$.

For each part of the search, we determine a pair of shortcuts with the shortest length and then we report the pair of shortcuts with the shortest encountered length. For a cycle with n vertices and k reflex vertices, the first part of the search takes $O(n)$ time, the second and third part take $O(kn)$ time each, and the forth part takes $O(k^2n)$. Therefore, we can find an optimal pair of shortcuts for convex cycles in $O(n)$ time and for non-convex cycles in $O(k^2n)$ time.

When the cycle is convex, we only perform the first part of the search, since every shortest pair of shortcuts with $\bowtie = ⧖ = \obslash = \oslash$ is optimal. We use this property—rather than convexity itself—to establish our result for convex cycles. This allows us to reuse the algorithm for convex cycles without modification for the first part of the search for non-convex cycles.

We present our algorithms for non-degenerate geometric cycles where the edges are line segments. Apart from the running time, our algorithmic results hold for more general edges, e.g., algebraic curves of constant degree. We assume, without loss of generality, that no two adjacent edges are co-linear: If two adjacent edges are co-linear, then we merge them into a single edge.

2.4.1 Algorithm for Convex Cycles

For a convex cycle C, we search for a shortest pair of shortcuts satisfying $\bowtie = ⧖ = \obslash = \oslash$. We begin with an arbitrary point p on the cycle C and compute three points q, r, and s such that $\bowtie = ⧖ = \obslash = \oslash$—regardless of whether pq and rs form a useful pair of shortcuts. The points q, r, and s exist for every point p along C and they are unique up to a (benign) degenerate case. Once we have an initial selection of points p, q, r, and s with $\bowtie = ⧖ = \obslash = \oslash$, we conceptually sweep p continuously along C maintaining the balance of the candidate cycles by moving q, r, and s appropriately. We show that q, r, and s move in the same direction as p while preserving their order along C. Thus, each endpoint traverses each edge of the n edges of C at most once throughout this process. This yields $O(n)$ events when discretizing the continuous sweep and, in between any two subsequent of these events, we can express the current diameter of $C + pq + rs$ as an algebraic function of constant degree, because the points p, q, r, and s remain on their containing edges. Therefore, we obtain an optimal shortcut for a convex cycle in $O(n)$ time.

We begin with an observation, illustrated in Figure 2.17, about the lengths of the sections into which the endpoints of the shortcuts divide the cycle when all candidate cycles are in balance.

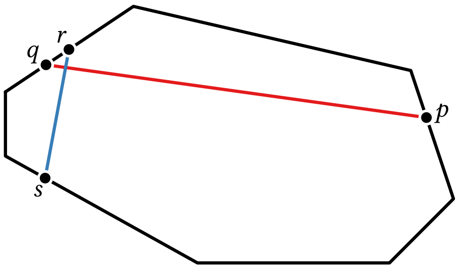


Figure 2.17: A convex cycle C with a pair of alternating shortcuts pq and rs with $\bowtie = ⧖ = \obslash = \oslash$. We have $\mathrm{diam}(C + pq + rs) = d(s, p)$ and $\mathrm{diam}(C + pq + rs) - \mathrm{diam}(C) = d(q, r)$.

Lemma 2.15. *Let pq, rs be a pair of alternating shortcuts for a geometric cycle C with $\bowtie = ⧖ = \obslash = \oslash$. The points p, r, q, and s divide C into four sections of lengths $a = d(p, r)$, $b = d(r, q)$, $c = d(q, s)$, and $d = d(s, p)$ with $a + b \leq c + d$ and $b + c \leq a + d$ such that*

$$\begin{aligned} d &= \frac{|C|}{4} + \frac{|pq| + |rs|}{2} = \mathrm{diam}(C + pq + rs) & a &= \frac{|C|}{4} + \frac{|pq| - |rs|}{2} \\ b &= \frac{|C|}{4} - \frac{|pq| + |rs|}{2} = \mathrm{diam}(C) - \mathrm{diam}(C + pq + rs) & c &= \frac{|C|}{4} + \frac{|rs| - |pq|}{2} \quad . \end{aligned}$$

Proof. Let pq, rs be a pair of alternating shortcuts for a geometric cycle C with $\bowtie = ⧖ = \obslash = \oslash$. Let $a = d(p, r)$, $b = d(r, q)$, $c = d(q, s)$, and $d = d(s, p)$. Recall that Lemma 2.6 implies

$$\obslash = \oslash \Leftrightarrow a = c + |pq| - |rs|, \quad \bowtie = ⧖ \Leftrightarrow a + c = b + d = |C|/2, \quad \bowtie = \oslash \Leftrightarrow b = a - |pq| \quad .$$

This implies $2a = a + c + |pq| - |rs| = |C|/2 + |pq| - |rs|$ and thus $2c = |C| - 2a = |C|/2 + |rs| - |pq|$. This yields $2b = 2a - 2|pq| = |C|/2 - |pq| - |rs|$ and thus $2d = |C| - 2b = |C|/2 + |pq| + |rs|$.

Since the red split cycle is a diametral cycle, we have $\obslash/2 = \mathrm{diam}(C + pq + rs)$. Furthermore, $\bowtie = \obslash$ implies $a + |rs| = d$ and, thus, $\obslash = a + |rs| + d$ implies $d = \obslash/2 = \mathrm{diam}(C + pq + rs)$. Since $b = |C|/2 - d$ and $\mathrm{diam}(C) = |C|/2$, we obtain $b = \mathrm{diam}(C) - \mathrm{diam}(C + pq + rs)$. □

Surprisingly, for every useful pair of alternating shortcuts pq and rs with $\bowtie = ⧖ = \obslash = \oslash$, we can read the new continuous diameter of $C + pq + rs$ from $d(s, p)$ and we can read the benefit of adding the shortcuts pq and rs to C from $d(r, q)$. Therefore, we may minimize $d(s, p)$ instead of $|pq| + |rs|$ when searching for a shortest pair of alternating shortcuts with $\bowtie = ⧖ = \obslash = \oslash$.

Consider a geometric cycle C and a fixed point p on C. We say a triple of points q, r, and s is *in balanced configuration with p* when the points p, r, q, and s appear counter-clockwise in this order along C with $d_{\mathrm{ccw}}(p, q) \leq |C|/2$ and $d_{\mathrm{ccw}}(r, s) \leq |C|/2$ such that $\bowtie = ⧖ = \obslash = \oslash$.

Lemma 2.16. *For every point p on a cycle C, there exist $q, r, s \in C$ that are in balance with p.*

Proof. Let p be a point on a geometric cycle C. Let $\bar{p}$ and $\bar{s}$ be the antipodal points of p and s, respectively, i.e., $d(p, \bar{p}) = d(s, \bar{s}) = |C|/2$. A pair of shortcuts pq and rs is alternating if and only if $|C|/4 \leq d_{\mathrm{ccw}}(s, p) \leq |C|/2$ and q, r appear in this order along the clockwise path from $\bar{p}$ to $\bar{s}$.

Let $a = d(p, r)$, $b = d(r, q)$, $c = d(q, s)$, and $d = d(s, p)$, as illustrated in Figure 2.18. An alternating pair of shortcuts pq, rs satisfies $\bowtie\, = ⧖$ if and only if $a + c = b + d = |C|/2$, i.e., $b = |C|/2 - d$. Thus, for fixed s and p, the position of q determines the position of r and vice versa.

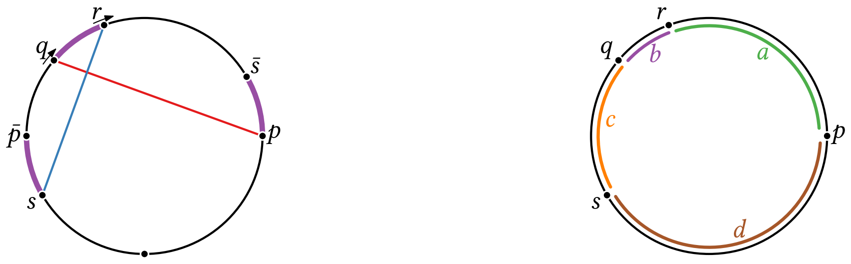


Figure 2.18: Locating q and r to balance $\bowtie\, = ⧖ = \oslash$ for fixed p and s.

Suppose we place s tentatively on C with $|C|/4 \le d_{\text{ccw}}(s, p) \le |C|/2$. We search for positions for q, r such that $\bowtie\, = ⧖ = \oslash$. We sweep q and r clockwise from $q = \bar{p}$ to $r = \bar{s}$ while maintaining $d(q, r) = |C|/2 - d(s, p)$, as illustrated in Figures 2.18 and 2.19. During the sweep, $\oslash$ weakly decreases. We argue that we have $\bowtie\, = ⧖ \ge \oslash$ when r reaches $\bar{s}$. If $\bowtie\, = ⧖ \le \oslash$ when q is at $\bar{p}$, then this implies that there exist positions for q, r such that $\bowtie\, = ⧖ = \oslash$. We use this to characterize the positions for s such that there exist q, r where pq, rs are alternating shortcuts with $\bowtie\, = ⧖ = \oslash$.

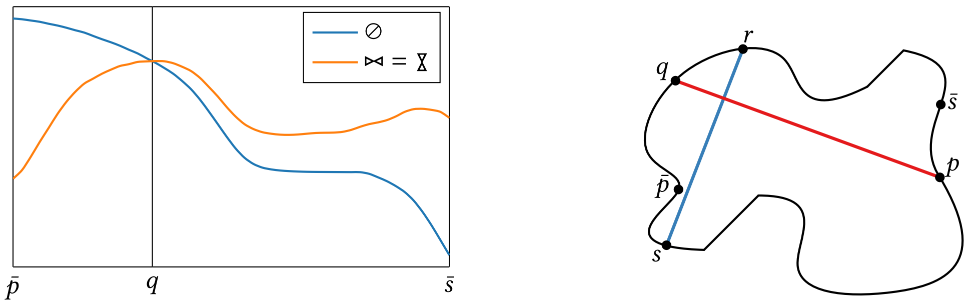


Figure 2.19: The points q, r where $\bowtie\, = ⧖ = \oslash$ for s, p on the geometric cycle shown on the right.

1. Suppose r lies at $\bar{s}$, as illustrated in Figure 2.20a. We argue that $⧖ = \bowtie\, \ge \oslash$.

   We have $c = d$, since $c = d(q, s) = d(s, \bar{s}) - d(q, \bar{s}) = |C|/2 - d(q, r) = d(s, p) = d$. This yields $⧖ = \bowtie\, \ge \oslash$, since $\bowtie\, = |pq| + |rs| + a + c = |pq| + |rs| + a + d \ge |rs| + a + d = \oslash$.

   Note that we have $⧖ = \bowtie\, = \oslash$ if and only if p and q coincide, i.e., $|pq| = 0$.

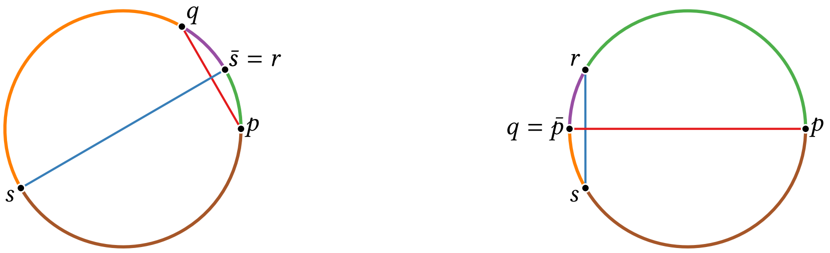
2. Suppose q lies at $\bar{p}$, as illustrated in Figure 2.20b.

   We have $b = c$, since $b = |C|/2 - d = d(\bar{p}, p) - d(s, p) = d(\bar{p}, s) = d(q, s) = c$.

   For each $\sim\, \in \{<, =, >\}$, we have $⧖ = \bowtie\, \sim \oslash$ if and only if

   $$d(\bar{p}, s) + |p\bar{p}| = c + |pq| \sim d = |C|/2 - b = |C|/2 - c = |C|/2 - d(\bar{p}, s) \ .$$

   This means we have $⧖ = \bowtie\, \le \oslash$ if and only if $d(\bar{p}, s) \le |C|/4 - |p\bar{p}|/2$.



(a) The case when r lies at $\bar{s}$.

(b) The case when q lies at $\bar{p}$.

Figure 2.20: The boundary cases when locating q and r to balance $\bowtie\, = \text{⧖} = \oslash$.

Let $\hat{p}$ be the unique position on the clockwise path from p to $\bar{p}$ with $d(\bar{p}, \hat{p}) = |C|/4 - |p\bar{p}|/2$. Notice that $0 \leq d(\bar{p}, \hat{p}) \leq |C|/4$, since $0 \leq |p\bar{p}| \leq |C|/2$ and notice that $\hat{p}$ is the farthest point from p along the cycle formed by the clockwise path from p to $\bar{p}$ and the line segment $\bar{p}p$. The points along the clockwise path from $\hat{p}$ to $\bar{p}$ are precisely the positions for s where there exist q, r such that pq, rs are alternating shortcuts with $\text{⧖} = \bowtie = \oslash$.

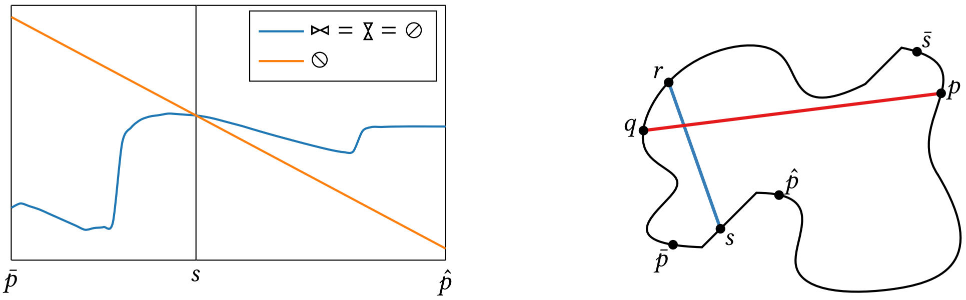


Figure 2.21: The points s, q, r that are in balance with p for the geometric cycle on the right.

We argue that $\bowtie\, = \text{⧖} = \oslash \geq \obslash$ when s is at $\hat{p}$ and that $\bowtie\, = \text{⧖} = \oslash \leq \obslash$ when s is at $\bar{p}$, as illustrated in Figure 2.21. Therefore, there exists a position for s along the clockwise path from $\hat{p}$ to $\bar{p}$ such that there exist q, r such that pq, rs are alternating shortcuts with $\bowtie\, = \text{⧖} = \obslash = \oslash$.

1. Suppose $s = \bar{p}$ and let q, r be such that $\text{⧖} = \bowtie = \oslash$, as illustrated in Figure 2.22a.

   We have $d = d(s, p) = d(\bar{p}, p) = |C|/2$. Since $\bowtie\, = \oslash$, we have $c + |pq| = d$. This implies

   $$\text{⧖} = \bowtie = \oslash \leq |C| = 2d = d + c + |pq| = \obslash \ .$$

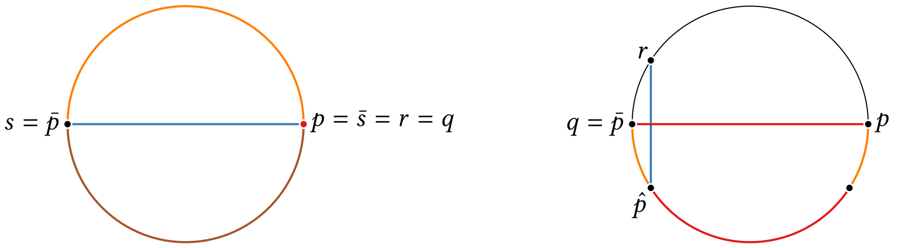
   The pair of shortcuts is never useful in this case, since pq is a line segment along C.

2. Suppose $s = \hat{p}$ and let q, r be such that $\text{⧖} = \bowtie = \oslash$, as illustrated in Figure 2.22b.

   Recall that $\hat{p}$ was the position for s where $\bowtie\, = \text{⧖} = \oslash$ when q lies at $\bar{p}$, which implies $b = c$, as argued above. Therefore, we have $\text{⧖} = \bowtie = \oslash \geq \obslash$, because

   $$\oslash = \bowtie = \text{⧖} = b + d + |pq| + |rs| \geq b + d + |pq| = c + d + |pq| = \obslash \ .$$

Therefore, there exist s, q, r such that pq, rs are alternating shortcuts with $\bowtie\, = \text{⧖} = \oslash = \obslash$. ☐



(a) The case when s is at $\bar{p}$.

(b) The case when s is at $\hat{p}$.

Figure 2.22: The boundary cases when locating a position for s with $\bowtie = ⧖ = \oslash = \obslash$. When $s = \bar{p}$ and C is not convex, q might not lie at p, but pq is part of an edge of C.

The three points s, q, r that are in balance with a point p on a geometric cycle C are unique up to the degeneracies described in Lemma 2.18. The following technical lemma summarizes some observations that simplify the proof of Lemma 2.18 and other arguments.

Lemma 2.17. *Let pq, rs and $p'q'$, $r's'$ be two pairs of alternating shortcuts. For $x \in \{p, q, r, s\}$, let $\delta_x = d_{\text{ccw}}(p, x') - d_{\text{ccw}}(p, x)$ and let $a = d(p, r)$, $b = d(r, q)$, $c = d(q, s)$, $d = d(s, p)$ and let $a' = d(p', r')$, $b' = d(r', q')$, $c' = d(q', s')$, $d' = d(s', p')$. Regardless of the signs of $\delta_p, \delta_q, \delta_r, \delta_s$,*

$$a' = a + \delta_r - \delta_p \qquad b' = b + \delta_q - \delta_r \qquad c' = c + \delta_s - \delta_q \qquad d' = d + \delta_p - \delta_s \quad ,$$

$$\bowtie = ⧖ \wedge \bowtie' = ⧖' \implies \delta_p + \delta_q = \delta_s + \delta_r \tag{2.1}$$
$$⧖ = \oslash \wedge ⧖' = \oslash' \implies 2\delta_r = |p'q'| - |pq| + \delta_p + \delta_q \tag{2.2}$$
$$\bowtie = \obslash \wedge \bowtie' = \obslash' \implies 2\delta_p = |r's'| - |rs| + \delta_r + \delta_s \quad . \tag{2.3}$$

Proof. The relations between a, b, c, d and a', b', c', d' follow from the definition of $\delta_p, \delta_q, \delta_r, \delta_s$.

If $\bowtie = ⧖$ and $\bowtie' = ⧖'$, then $b + d = |C|/2 = b' + d'$. This implies $\delta_p + \delta_q = \delta_s + \delta_r$, since

$$\begin{aligned} \delta_p + \delta_q &= \delta_p + \delta_q + (b + d) - (b' + d') \\ &= \delta_p + \delta_q - (d' - d) - (b' - b) \\ &= \delta_p + \delta_q - (\delta_p - \delta_s) - (\delta_q - \delta_r) = \delta_s + \delta_r \quad . \end{aligned}$$

If $⧖ = \oslash$ and $⧖' = \oslash'$, then we have $0 = ⧖ - \oslash = |pq| + b - a$ and $0 = ⧖' - \oslash' = |p'q'| + b' - a'$. This implies $2\delta_r = |p'q'| - |pq| + \delta_p + \delta_q$, because

$$\begin{aligned} 2\delta_r &= 2\delta_r + (⧖' - \oslash') - (⧖ - \oslash) \\ &= 2\delta_r + (|p'q'| + b' - a') - (|pq| + b - a) \\ &= 2\delta_r + |p'q'| - |pq| + (b' - b) - (a' - a) \\ &= 2\delta_r + |p'q'| - |pq| + (\delta_q - \delta_r) - (\delta_r - \delta_p) \\ &= |p'q'| - |pq| + \delta_p + \delta_q \quad . \end{aligned}$$

Analogously, we show that if $\bowtie = \obslash$ and $\bowtie' = \obslash'$, then $2\delta_p = |r's'| - |rs| + \delta_r + \delta_s$. □

Lemma 2.18. *Let C be a geometric cycle. If q, r, s and q', r', s' are in balance with $p \in C$ where $d(s', p) \leq d(s, p)$, then one of the cases in Figure 2.23 applies and $|pq'| + |r's'| \leq |pq| + |rs|$.*

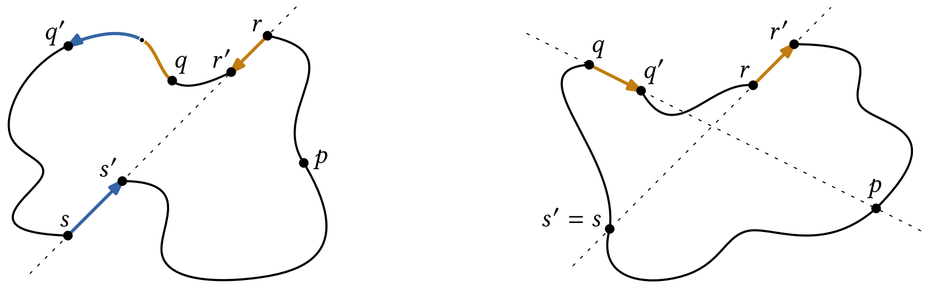
Proof. Suppose there exist two triples s, q, r and s', q', r' that are in balanced configuration with a point p along a geometric cycle C. Furthermore, suppose $d(s', p) \leq d(s, p)$.

Let $\delta_x = d_{\text{ccw}}(p, x') - d_{\text{ccw}}(p, x)$ for $x \in \{q, r, s\}$. We place p' at p. Lemma 2.17 yields

$$a' = a + \delta_r \qquad b' = b + \delta_q - \delta_r \qquad c' = c - \delta_q + \delta_s \qquad d' = d - \delta_s \quad .$$

We have $\delta_s \geq 0$, because $d_{\text{cw}}(s', p) = d(s', p) \leq d(s, p) = d_{\text{cw}}(s, p)$. Furthermore, Lemma 2.17 implies $\delta_q = \delta_s + \delta_r$ and $2\delta_r = |pq'| - |pq| + \delta_q$ and $|rs| = |r's'| + \delta_r + \delta_s$.

We distinguish two cases depending on whether q' lies clockwise or counter-clockwise of q. Each case corresponds to one of the two degenerate constellations depicted in Figure 2.23.



(a) The counter-clockwise case $\delta_q > 0$.

(b) The clockwise case $\delta_q < 0$.

Figure 2.23: The two cases when there exist two triples s, q, r and s', q', r' that are in balance with a point p along a geometric cycle C with $d(s', p) \leq d(s, p)$. If q' lies counter-clockwise of q, i.e., $\delta_q > 0$, then $|rs| = d(r, r') + |r's'| + d(s, s')$ and the cycle C contains the line segments rr' and ss', as depicted in Figure 2.23a. If q' lies clockwise of q, i.e., $\delta > 0$, then $s = s'$ and $|r's| = d(r', r) + |rs|$ and $|pq| = |pq'| + d(q', q)$ and C contains the line segments rr' and qq' that have equal length, as depicted in Figure 2.23b.

1. Suppose $\delta_q > 0$, i.e., q' lies counter-clockwise of q. This means $\delta_q = d(q, q')$.

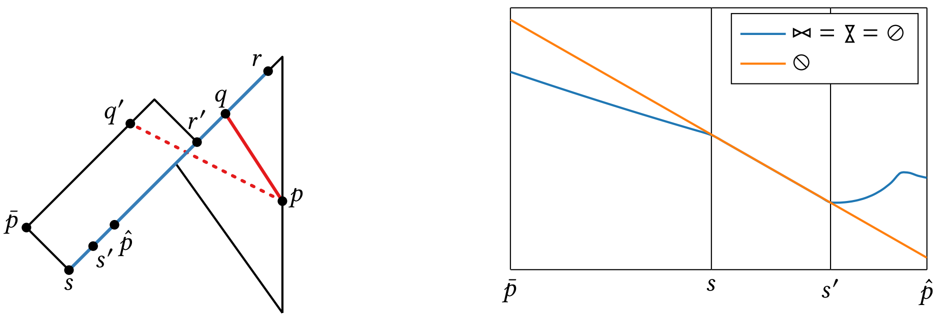
   Then, we have $\delta_r \geq 0$, i.e., r' lies counter-clockwise of r and, thus, $\delta_r = d(r, r')$, since

   $$2\delta_r = |pq'| - |pq| + \delta_q = |pq'| - |pq| + d(q, q') \geq -d(q, q') + d(q, q') = 0 \quad .$$

   This implies $|rs| = |r's'| + \delta_r + \delta_s = d(r, r') + |r's'| + d(s', s)$. Therefore, r, r', s, and s' are co-linear and the line segment rs consists of the counter-clockwise path from r to r', the line segment $r's'$, and the counter-clockwise path from s to s', as in Figures 2.23a and 2.24.

   We argue $|pq'| + |r's'| \leq |pq| + |rs|$. Recall that $\text{⧗} = \oslash$ implies $|pq| = a - b$ and $\text{⧗}' = \oslash'$ implies $|p'q'| = a' - b'$. Since $\delta_q = \delta_r + \delta_s$ implies $b' = b + \delta_q - \delta_r = b + \delta_s$, we obtain

   $$\begin{aligned} |pq'| + |r's'| &= |pq'| + |rs| - \delta_r - \delta_s = a' - b' + |rs| - \delta_r - \delta_s \\ &= (a + \delta_r) - (b + \delta_s) + |rs| - \delta_r - \delta_s = a - b + |rs| - 2\delta_s \\ &= |pq| + |rs| - 2\delta_s \leq |pq| + |rs| \quad . \end{aligned}$$



(a) The positions of the endpoints along the cycle.   (b) A comparison of ⋈ = ⧖ = ⊘ and ⦸.

Figure 2.24: A geometric cycle C with a point $p \in C$ such that there is more than one position for s that leads to a triple s, q, r that is in balance with p.

2. Suppose $\delta_q < 0$, i.e., q' lies counter-clockwise of q. This means $\delta_q = -d(q, q')$.

   We have $\delta_r < 0$, since $\delta_r \le \delta_r + \delta_s = \delta_q < 0$. Hence, $\delta_r = -d(r, r')$.

   We have $|pq| = |pq'| + d(q, q')$ and $|r's'| = d(r', r) + |rs| + d(s, s')$, because the inequalities

$$\begin{aligned} 0 = -d(q,q') + d(q,q') &\le |pq'| - |pq| + d(q,q') = |pq'| - |pq| - \delta_q \\ &= \obslash' - \obslash = \oslash' - \oslash = |r's'| - |rs| + \delta_r - \delta_s \\ &= |r's'| - |rs| - d(r,r') - d(s,s') \\ &\le d(r,r') + d(s,s') - d(r,r') - d(s,s') = 0 \end{aligned}$$

   are attained with equality. This implies $\oslash' = \oslash$, since $d(r', r) = -\delta_r$, and, thus

$$\oslash' = |r's'| + a' + d' = |rs| + \delta_s - \delta_r + (a + \delta_r) + (d - \delta_s) = |rs| + a + d = \oslash \ .$$

   Therefore, we have $\delta_s = 0$ and, thus $s = s'$, because $|pq'| - |pq| = -d(q, q') = \delta_q$ and $|r's'| - |rs| = d(r', r) + d(s, s') = \delta_s - \delta_r$ and $a' - a = \delta_r$ and $c' - c = \delta_s - \delta_q$ and, thus,

$$\begin{aligned} 0 = \oslash' - \oslash = \bowtie' - \bowtie &= (|pq'| + |r's'| + a' + c') - (|pq| + |rs| + a + c) \\ &= (|pq'| - |pq|) + (|r's'| - |rs|) + (a' - a) + (c' - c) \\ &= (\delta_q) + (\delta_s - \delta_r) + (\delta_r) + (\delta_s - \delta_q) = 2\delta_s \ . \end{aligned}$$

   We conclude $|pq'| + |r's'| = |pq| + |rs|$, as $-d(q, q') = \delta_q = \delta_r + \delta_s = \delta_r = -d(r, r')$ and, thus, $|pq'| + |r's'| = |pq| - d(q, q') + |rs| + d(r, r') + d(s, s') = |pq| + |rs|$.

Therefore, we have one of the constellations in Figure 2.23 and $|p'q'| + |r's'| \le |pq| + |rs|$. □

Lemma 2.18 suggests that we handle the cases where the points that are in balance with p are not unique, as follows. First, if there are several positions for s that lead to balanced triples with p, then we pick the one closest to p, since this will lead to the shortest pair of shortcuts. Second, if there are several positions for q, then we pick, say, the position with the least clockwise distance to p, since all the choices for q lead to pairs of shortcuts with the same length. Therefore, we assume in the following, without loss of generality, that the balanced triple for p is unique.

Corollary 2.19. *For a geometric cycle C with n vertices and a point $p \in C$, it takes $O(n)$ time to compute three points $q, r, s \in C$ that are in balanced configuration with $p \in C$.*

Proof. Let p be a point on a geometric cycle C with n vertices. We compute the counter-clockwise distance from p to each vertex of C. This takes $O(n)$ time and supports constant time queries for the network distance between any two points along the edges of C. This also allows us to determine the value of $\bowtie$, $⧖$, $\obslash$, and $\oslash$ for any pair of shortcuts pq, rs in constant time.

The positions of s and q imply the position of r, due to the constraint $\bowtie = ⧖$. For a fixed position of s, all positions of q that lead to a triple s, q, r with $\bowtie = ⧖ = \oslash$ form a possibly empty interval along C. The positions for q clockwise of this interval lead to $\bowtie = ⧖ > \oslash$ and the positions for q counter-clockwise of this interval lead to $\bowtie = ⧖ < \oslash$, as illustrated in Figure 2.19. Therefore, for a fixed position of s, we can find a position of q with $\bowtie = ⧖ = \oslash$ in $O(\log n)$ time using binary search, if it exists. We detect in constant time if a suitable position for q exists.

The positions of s such that there exist q and r with $\bowtie = ⧖ = \oslash$ form a non-empty interval along C. Within this interval there exists a line segment in C containing all positions for s that lead to a balanced triple. All positions for s clockwise of this line segment lead to $\bowtie = ⧖ = \oslash < \obslash$ and all positions counter-clockwise of this line segment lead to $\bowtie = ⧖ = \oslash > \obslash$, as illustrated in Figure 2.21. Therefore, we can find a suitable position for s in $O(\log^2 n)$ time with a binary search that performs in each step a binary search for a position for q with $\bowtie = ⧖ = \oslash$. ☐

Lemma 2.20. *We sweep a point p along a cycle C while maintaining the points s, q, r that are in balance with p. As p traverses C, each of q, r, and s enter each edge at most once through a vertex.*

Proof. Let pq, rs and $p'q'$, $r's'$ be two pairs of shortcuts with $p \neq p'$ and $d_{\text{ccw}}(p, p') = d(p, p')$ such that $\bowtie = ⧖ = \oslash = \obslash$ and $\bowtie' = ⧖' = \oslash' = \obslash'$. We argue that either all endpoints move in the same direction as p sweeps to p' or all endpoints remain on their containing edge.

For $x \in \{p, q, r, s\}$, let $\delta_x = d_{\text{ccw}}(p, x') - d_{\text{ccw}}(p, x)$. In particular, we have $\delta_p = d(p, p') > 0$.

We argue that $\delta_s > 0$. By symmetry, Lemmas 2.16 and 2.18 imply that for every position of s along C there is a unique point p such that we can find $q, r \in C$ that lead to a balance in the four candidates for diametral cycles in $C + pq + rs$. Therefore, s cannot remain stationary or change direction while p sweeps along C, i.e., $\delta_s \neq 0$. If s moves clockwise ($\delta_s < 0$) while p moves counter-clockwise ($\delta_p > 0$), then we eventually reach positions s'' and p'' such that $d(p'', s'') = |C|/2$, i.e., s'' is the farthest point $\bar{p}''$ from p'' along C. In the proof of Lemma 2.16, we have seen that this means every q'', r'' with $\bowtie'' = ⧖'' = \oslash''$ and $\obslash'' = \oslash''$ satisfy $\bowtie'' = ⧖'' < \obslash'' = \oslash''$. This contradicts the existence of a balanced triple for p''. Therefore, s must move in the same direction as p, i.e., $\delta_s > 0$ and, thus, s enters each edge at most once through a vertex.

Regardless of the signs of δ_r and δ_q, the changes in the section lengths are

$$a' = a + \delta_r - \delta_p \qquad b' = b + \delta_q - \delta_r \qquad c' = c + \delta_s - \delta_q \qquad d' = d + \delta_p - \delta_s \ ,$$

and, by Lemma 2.17, the balances $\bowtie = ⧖ = \obslash = \oslash$ and $\bowtie' = ⧖' = \obslash' = \oslash'$ imply $\delta_p + \delta_q = \delta_s + \delta_r$ and $2\delta_p = \delta_s + \delta_r + |r's'| - |rs|$ and $2\delta_r = \delta_q + \delta_p + |p'q'| - |pq|$.

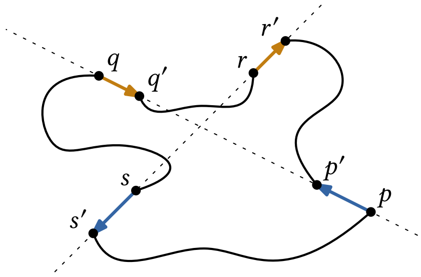


Figure 2.25: The degenerate case where we sweep a point p counter-clockwise to p' along a geometric cycle C and the point r moves clockwise to r' where s, q, r and $s', q,' , r'$ are the triples of points that are in balance with p and p', respectively. We have $|pq| = d(p, p')+|p'q'|+d(q', q)$ and $|r's'| = d(r', r)+|rs|+d(s, s')$ with $d(p, p') = d(s, s')$ and $d(r, r') = d(q, q')$. This means $|p'q'| + |r's'| = |pq| + |rs|$.

We distinguish the following two cases: In the first case, r moves in the same direction as p or r remains stationary, i.e., $\delta_r \geq 0$, and all endpoints move in the same direction as p sweeps to p'. In the second case, r moves in a different direction as p, i.e., $\delta_r < 0$, and all endpoints remain on their containing edge as p sweeps to p', as illustrated in Figure 2.25.

1. Suppose $\delta_r \geq 0$. Then $\delta_r = d(r, r')$ and, thus, $\delta_q \geq 0$, since

$$\begin{aligned} 2\delta_q &= 2\delta_r + 2\delta_s - 2\delta_p && (\delta_p + \delta_q = \delta_r + \delta_s) \\ &= \delta_r + \delta_s + d(r, r') + d(s, s') - 2\delta_p && (\delta_r = d(r, r') \text{ and } \delta_s = d(s, s')) \\ &\geq \delta_r + \delta_s + |r's'| - |rs| - 2\delta_p && (d(r, r') + d(s, s') \geq |r's'| - |rs|) \\ &= 2\delta_p - 2\delta_p && (\delta_r + \delta_s + |r's'| - |rs| = 2\delta_p) \\ &= 0 \ . \end{aligned}$$

   Therefore, s, q, and r move in the same direction as p and—since none of these endpoints may pass each other—each endpoint enters each edge of C at most once through a vertex.

2. Suppose $\delta_r < 0$. This means, we have $\delta_r = -d(r, r')$.

   We argue that q moves in the same direction as r with $\delta_q \leq \delta_r < 0$. By definition, we have either $\delta_q = d(q, q')$ or $\delta_q = -d(q, q')$. The latter applies, since $\delta_q < d(q, q')$, because

$$\begin{aligned} \delta_q &= \delta_q + d(p, p') - d(p, p') - d(q, q') + d(q, q') && \text{(adding zeros)} \\ &= \delta_q + \delta_p - d(p, p') - d(q, q') + d(q, q') && (d(p, p') = \delta_p) \\ &\leq \delta_q + \delta_p + |p'q'| - |pq| + d(q, q') && (-d(p, p') - d(q, q') \leq |p'q'| - |pq|) \\ &= 2\delta_r + d(q, q') && (\delta_p + \delta_q + |p'q'| - |pq| = 2\delta_r) \\ &< d(q, q') \ . && (\delta_r < 0) \end{aligned}$$

   This implies $\delta_q \leq \delta_r$, as $d(q, q') = -\delta_q$ yields $2\delta_q \leq \delta_q + 2\delta_r + d(q, q') = \delta_q + 2\delta_r - \delta_q = 2\delta_r$.

We argue that s moves in the same direction as p with $0 < \delta_p \leq \delta_s$. We already know that $d(s, s') = \delta_s > 0$ and we obtain $\delta_p \leq \delta_s$, since $\delta_r = -d(r, r')$ leads to

$$2\delta_p = \delta_r + \delta_s + |r's'| - |rs| \leq \delta_r + \delta_s + d(r, r') + d(s, s') = \delta_r + \delta_s - \delta_r + \delta_s = 2\delta_s \ .$$

Now $\delta_p + \delta_q = \delta_s + \delta_r$ and $\delta_q \leq \delta_r$ and $\delta_p \leq \delta_s$ imply $\delta_p = \delta_s$ and $\delta_q = \delta_r$, since

$$\delta_p + \delta_q \leq \delta_p + \delta_r \leq \delta_s + \delta_r = \delta_p + \delta_q \ .$$

This leads to a degenerate constellation where p, p', q, q' are co-linear and r, r', s, s' are co-linear in a way that prohibits q, and r to enter another edge through a vertex.

We obtain $|pq| = d(p, p') + |p'q'| + d(q', q)$, since $2\delta_q = 2\delta_r = \delta_p + \delta_q + |p'q'| - |pq|$ implies

$$|pq| = \delta_p + |p'q'| - \delta_q = d(p, p') + |p'q'| + d(q', q) \ .$$

We obtain $d(r', r) + |rs| + d(s, s') = |r's'|$, since $2\delta_s = 2\delta_p = \delta_r + \delta_s + |r's'| - |rs|$ implies

$$d(r', r) + |rs| + d(s, s') = -\delta_r + |rs| + \delta_s = |r's'| \ .$$

Therefore, we have the degenerate situation in Figure 2.25 where C contains the line segments pp', qq', rr', and ss'. Since we merged all adjacent co-linear edges, this means that all endpoints remain on their containing edges as p sweeps to p' in this case.

In conclusion, as p sweeps the entire cycle C once, each of the points s, q, r that are in balance with p (and the point p itself) enter each edge of C at most once through a vertex. ☐

Theorem 2.21. *For a convex cycle C with n vertices, it takes $O(n)$ time to determine a pair of shortcuts pq, rs that minimizes the continuous diameter of the augmented cycle $C + pq + rs$.*

Proof. We pick an arbitrary point p along C and determine three points s, q, r that are in balanced configuration with p. This takes $O(n)$ time, as argued in Corollary 2.19. Conceptually, we sweep p continuously along C while maintaining s, q, r such that $\bowtie = \text{⧖} = \oslash = \obslash$ and monitoring the distance $d(s, p)$ that matches the diameter of $C + pq + rs$, as argued in Lemma 2.15.

To discretize the sweep we proceed as follows. In each step, we identify the four edges that would contain p, q, r, and s next, if p was to sweep continuously along C: for each $x \in \{p, q, r, s\}$, we calculate how far the other endpoints would move under the assumption that x is the first point to hit a vertex. Since all points move in the same direction as p, an edge e will never host an endpoint x in any subsequent step, once x has left e. Therefore, the four edges e_p, e_q, e_r, and e_s that contain p, q, r, and, s, respectively, change $O(n)$ times. For each set of edges, we determine a pair of alternating shortcuts p^*q^*, r^*s^* with $\bowtie^* = \text{⧖}^* = \obslash^* = \oslash^*$ that is optimal among all pairs of shortcuts with $x^* \in e_x$ for all $x \in \{p, q, r, s\}$. This takes constant time, since we can express the value of $d(s, p) = \mathrm{diam}(C + pq + rs)$ between subsequent events of the sweep as an algebraic function of constant degree in terms of the position of p along e_p.

Therefore, we obtain an optimal pair of shortcuts for a convex cycle C in $O(n)$ time. ☐

2.4.2 Algorithm for Non-Convex Cycles

As explained at the beginning of this section, our search for an optimal pair of shortcuts for a geometric cycle consists of four parts: The search for the shortest pair of shortcuts where (1) none of the shortcuts are stuck at a reflex vertex ($\bowtie = ⧖ = \obslash = \oslash$), (2,3) one of the shortcuts is stuck at a reflex vertex ($\bowtie = ⧖ = \oslash > \obslash$, $\bowtie = ⧖ = \obslash > \oslash$), and (4) both of the shortcuts are stuck at a reflex vertex ($\bowtie = ⧖ > \obslash, \oslash$). The algorithm for convex cycles from Theorem 2.21 already handles the first part of this search, because—instead of relying on convexity itself—it relies on the balance of the four candidate cycles ($\bowtie = ⧖ = \obslash = \oslash$) that holds for convex cycles.

We begin with the case where the red shortcut is stuck at a reflex vertex ($\bowtie = ⧖ = \oslash > \obslash$). We describe how to find the best shortcut with p at a reflex vertex and similarly how to find the best shortcut with q stuck at a reflex vertex. For a geometric cycle C with n vertices and k reflex vertices, this takes $O(n)$ time per reflex vertex and, thus, $O(kn)$ total time. The treatment for the case when the blue shortcut is stuck is symmetric.

Lemma 2.22. *Let C be a cycle with n vertices. For every $p \in C$, it takes $O(n)$ time to determine $s, q, r \in C$ that minimize $|pq| + |rs|$ for all alternating shortcuts pq, rs with $\bowtie = ⧖ = \oslash > \obslash$.*

Proof. Let p be a point along a geometric cycle C. Let $\bar{p}$ be the farthest point from p along C and let $\hat{p}$ be the farthest point from p along the cycle formed by the clockwise path from p to $\bar{p}$ and the line segment $p\bar{p}$, i.e., $d(\bar{p}, \hat{p}) = |C|/4 - |p\bar{p}|/2$. We know from the proof of Lemma 2.16 that the clockwise path from $\hat{p}$ to $\bar{p}$ consists of all positions for s where there exist q, r such that the shortcuts pq, rs are alternating and satisfy $\bowtie = ⧖ = \oslash$. Due to the constraint $⧖ = \bowtie = \oslash$, the position of s implies the positions of q and r and these positions are unique up to the degeneracies in Figure 2.23b that lead to pairs of shortcuts with the same length. According to Lemma 2.18, there are two points $s_=$ and $s'_=$ along the clockwise path from $\hat{p}$ to p such that the clockwise path from $s_=$ to $s'_=$ consists of all positions for s where we achieve $\bowtie = ⧖ = \oslash = \obslash$. We have $s_= = s'_=$, except for the degenerate case depicted in Figure 2.23a. Furthermore, we achieve $\bowtie = ⧖ = \oslash \leq \obslash$ when $s = \bar{p}$ and we achieve $\bowtie = ⧖ = \oslash \geq \obslash$ when $s = \hat{p}$. Therefore, the positions for s where there exist q, r such that pq and rs are alternating shortcuts with $\bowtie = ⧖ = \oslash > \obslash$ form the clockwise path from $\hat{p}$ to $s_=$, excluding $s_=$ itself, as illustrated in Figure 2.24, where $s_=$ is s' and $s'_=$ is s.

We sweep s counter-clockwise from $s_=$ to $\hat{p}$. We argue that q and r move counter-clockwise throughout the sweep. This means that the sequence of edges containing s, q, r has length $O(n)$. When s sweeps along an edge e_s such that q lies on an edge e_q and r lies on an edge e_r, then we can describe $|pq| + |rs|$ such that $⧖ = \bowtie = \oslash$ as an algebraic function of low degree and determine its minimum in constant time. Therefore, it takes $O(n)$ time to locate the shortest pair of alternating shortcuts pq and rs with $\bowtie = ⧖ = \oslash > \obslash$ for a fixed position of p.

It remains to show that q, r move counter-clockwise, as s moves counter-clockwise from $s_=$ to $s'_=$. Suppose s, q, r and s', q', r' are two positions with $d(p, s) \geq d(p, s')$, such that $\bowtie = ⧖ = \oslash$ and $\bowtie' = ⧖' = \oslash'$, respectively. Let $a = d(p, r)$, $b = d(r, q)$, $c = d(q, s)$, $d = d(s, p)$, and let $a' = d(p, r')$, $b' = d(r', q')$, $c' = d(q', s')$, $d' = d(s', p)$. For $x \in \{s, q, r\}$, let $\delta_x = d_{\text{ccw}}(p, x') - d_{\text{ccw}}(p, x)$. The points s, q, r move counter-clockwise to s', q', r' if and only if $\delta_s, \delta_q, \delta_r \geq 0$.

The choice $d(p, s) \geq d(p, s')$ implies $\delta_s = d(s, s') > 0$. We have $\bowtie = \oslash$ and $\bowtie' = \oslash'$ and, thus, $2\delta_s = \delta_q + |pq| - |pq'|$, by Lemma 2.17. We derive that $\delta_q > 0$, since the assumption $\delta_q \leq 0$ implies

$\delta_q = -d(q, q')$ and leads to the contradiction $0 < 2\delta_s = \delta_q + |pq| - |pq'| \leq \delta_q + d(q, q') = 0$.

We have $\text{⧖} = \oslash$ and $\text{⧖}' = \oslash'$ and, thus, $2\delta_r = \delta_q + |pq'| - |pq|$, by Lemma 2.17. We derive that $\delta_r \geq 0$, since $0 < \delta_q = d(q, q')$ and, thus, $2\delta_r = \delta_q + |pq'| - |pq| \geq \delta_q - d(q, q') = 0$.

Therefore, q and r move counter-clockwise as s sweeps in counter-clockwise direction along the counter-clockwise path from $s_=$ to $\hat{p}$. As argued above, this means that the claim follows. □

Corollary 2.23. *Consider a geometric cycle C with n vertices and k reflex vertices. It takes $O(kn)$ time to determine a shortest pair of alternating shortcuts pq, rs such that $\bowtie = \text{⧖} = \oslash > \obslash$.*

Proof. We know that if pq, rs is a shortest pair of alternating shortcuts with $\bowtie = \text{⧖} = \oslash > \obslash$, then the red shortcut pq has one endpoint at a reflex vertex of C. For each reflex vertex v of C, we search for a shortest pair of alternating shortcuts pq, rs with $\bowtie = \text{⧖} = \oslash > \obslash$ and $p = v$. This takes $O(n)$ time per reflex vertex. Analoguously it takes $O(n)$ time per reflex vertex to find a shortest pair of alternating shortcuts pq, rs with $\bowtie = \text{⧖} = \oslash > \obslash$ and $q = v$. Therefore, it takes $O(kn)$ time to find a shortest alternating pair of shortcuts pq, rs such that $\bowtie = \text{⧖} = \oslash > \obslash$. □

Theorem 2.24. *Consider a geometric cycle C with n vertices and k reflex vertices. It takes $O(k^2 n)$ time to determine a shortest pair of alternating shortcuts pq, rs such that $\bowtie = \text{⧖} > \oslash, \obslash$.*

Proof. If pq, rs is a shortest pair of alternating shortcuts with $\bowtie = \text{⧖} > \oslash, \obslash$, then both shortcuts have an endpoint at a reflex vertex of C. Suppose u, v are reflex vertices of C with $|C|/4 \leq d_{\text{ccw}}(u, v) \leq |C|/2$. We can find the shortest alternating pair of shortcuts pq, rs with $s = u$ and $p = v$ such that $\bowtie = \text{⧖} > \oslash, \obslash$ in $O(n)$ time. This is because any position for q determines the position for r, since $d(q, r) = b = |C|/2 - d = |C|/2 - d(u, v)$. Hence, it is sufficient to sweep q and r from $q = \bar{v}$ to $r = \bar{u}$ and report the shortest encountered pair with $\bowtie = \text{⧖} > \obslash, \oslash$. Analogously, it takes $O(n)$ time to find the shortest alternating pair of shortcuts pq, rs such that $\bowtie = \text{⧖} > \oslash, \obslash$ when $u \in \{s, r\}$ and $v \in \{p, q\}$. Since there are $\binom{k}{2}$ pairs of reflex vertices, it takes $O(k^2 n)$ time to determine a shortest pair of alternating shortcuts pq, rs such that $\bowtie = \text{⧖} > \oslash, \obslash$. □

In conclusion, we have seen that for every geometric cycle there is an optimal pair of shortcuts that is a shortest pair of alternating shortcuts with $\bowtie = \text{⧖} \geq \oslash, \obslash$. In this section, we argued that it takes $O(n)$ time to find a shortest pair of shortcuts with $\bowtie = \text{⧖} = \oslash = \obslash$, it takes $O(kn)$ times to find a shortest pair of shortcuts with $\bowtie = \text{⧖} = \oslash > \obslash$ or $\bowtie = \text{⧖} = \obslash > \oslash$, and it takes $O(k^2 n)$ time to find a shortest pair of shortcuts with $\bowtie = \text{⧖} > \oslash, \obslash$. Therefore, it takes $O(k^2 n)$ time to find an optimal pair of shortcuts for a geometric cycle with n vertices and k reflex vertices.

Theorem 2.25. *For a non-convex geometric cycle C with n vertices and k reflex vertices, it takes $O(k^2 n)$ time to determine a pair of shortcuts pq, rs for C that minimizes $\operatorname{diam}(C + pq + rs)$.* □

Recall that we have $d(s, p) = \operatorname{diam}(C + pq + rs)$ for every pair of alternating shortcuts pq, rs with $\bowtie = \text{⧖} = \oslash = \obslash$. However, we have $\operatorname{diam}(C + pq + rs) > d(s, p)$ when $\bowtie = \text{⧖} > \oslash, \obslash$, since

$$2 \operatorname{diam}(C + pq + rs) = \bowtie = a + c + |pq| + |rs| > d + c + |pq| > 2d = 2d(s, p) \ .$$

This means that we cannot minimize the continuous diameter of a geometric cycle C by minimizing $d(s, p)$ instead of $|pq| + |rs|$ among all alternating shortcuts pq, rs for C with $\bowtie = \text{⧖} \geq \oslash, \obslash$.

3 Tree Shortcuts

We are given a geometric tree T in the plane. We wish to find a shortcut pq for T that minimizes the continuous diameter of the resulting tree $T + pq$, as illustrated in Figure 3.1.

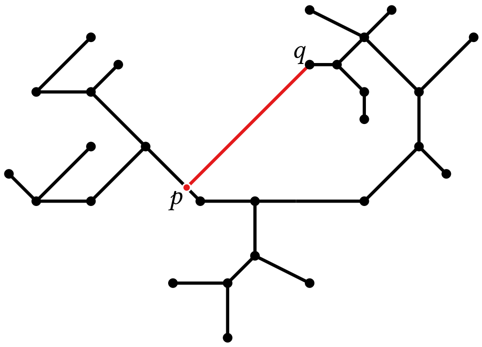


Figure 3.1: A shortcut for a geometric tree.

We require the following definitions to describe our results. The *backbone* of a tree T is the intersection of all diametral paths of T; we denote the backbone of T by $\mathcal{B}$. The *absolute center* of a geometric tree T is the unique point $c \in T$ that minimizes the largest network distance from c, i.e., $\max_{q \in T} d_T(c, q) = \min_{p \in T} \max_{q \in T} d_T(p, q)$. The absolute center c is the midpoint of every diametral path in T and, thus, c always lies on the backbone $\mathcal{B}$ of T.

We present the following structural results about optimal shortcuts for geometric trees in Section 3.1. A shortcut pq is *useful* for a geometric tree T when augmenting T with pq decreases the continuous diameter. A geometric tree T admits a useful shortcut if and only if its backbone $\mathcal{B}$ is neither a line segment nor a single point. Every geometric tree T has an optimal shortcut pq with $p, q \in \mathcal{B}$ such that the absolute center c lies on the path from p to q in T.

We determine an optimal shortcut for a geometric tree T with n vertices in $O(n \log n)$ time. Conceptually, we slide a candidate shortcut pq along the backbone of T while balancing the diametral paths in $T + pq$. To implement this approach, we discretize this movement such that the shortcut jumps from one change in the diametral paths to the next. The diametral pairs in $T + pq$ guide this search as each rules out a better shortcut in some direction. In Section 3.3, we assemble the building blocks of the algorithm and explain how the diametral pairs guide our search. In Section 3.4, we describe the continuous algorithm and show that it produces an optimal shortcut. In Section 3.5, we discretize this algorithm and introduce some modifications to the continuous algorithm that are necessary to achieve the desired running time.

3.1 Usefulness

We say a shortcut pq is *useful* for T when $\operatorname{diam}(T + pq) < \operatorname{diam}(T)$, we say pq is *indifferent* for T when $\operatorname{diam}(T + pq) = \operatorname{diam}(T)$, and we say pq is *useless* for T when $\operatorname{diam}(T + pq) > \operatorname{diam}(T)$. In the discrete setting, every shortcut is useful or indifferent, as the discrete diameter only considers vertices of T. In the continuous setting, a shortcut may be useless for T, since the points on the shortcut pq matter as well, as exemplified in Figure 3.2.

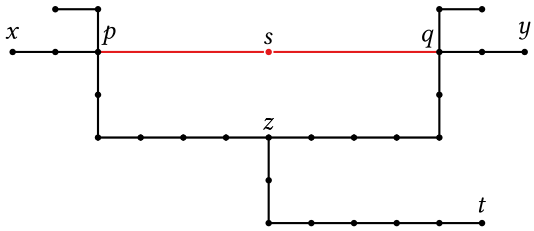


Figure 3.2: A geometric tree T with a shortcut pq that is useless for T. Each edge has a unit weight and $|pq| = 8$. The continuous diameter increases when we augment T with pq, since $\operatorname{diam}(T) = d_T(x, y) = 16$ and $\operatorname{diam}(T + pq) = d_{T+pq}(s, t) = 17$.

For two points $u, v \in T$, we say that a shortcut pq is *useful* for the unordered pair $\{u, v\}$ when $d_{T+pq}(u, v) < d_T(u, v)$, and we say that pq is *indifferent* for $\{u, v\}$ when $d_{T+pq}(u, v) = d_T(u, v)$.

We say a shortcut pq is *useful* for the ordered pair u, v when $d_T(u, p) + |pq| + d_T(q, v) < d_T(u, v)$, we say pq is *indifferent* for u, v when $d_T(u, p) + |pq| + d_T(q, v) = d_T(u, v)$, and we say that pq is *useless* for u, v when $d_T(u, p) + |pq| + d_T(q, v) > d_T(u, v)$. If pq is useful for u, v then the shortest path from u to v in $T + pq$ travels from u to p in T, then along the shortcut to q, and then from q to v in T. Order matters: if pq is useful for the ordered pair u, v then pq is useless for v, u.

For some trees, we cannot reduce the continuous diameter with a single shortcut. For instance, every shortcut pq for the tree T in Figure 3.3 is either useless or indifferent for T, because pq would be indifferent for at least one of the diametral pairs $\{x, y\}$, $\{x, z\}$, or $\{y, z\}$ of T.

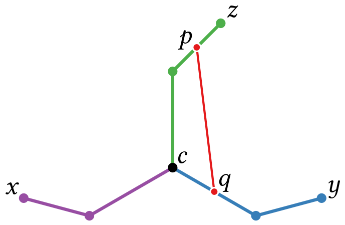


Figure 3.3: A geometric tree T whose continuous diameter cannot be reduced with a single shortcut. Each pair of the three leaves x, y, and z is diametral, as $d_T(x, c) = d_T(y, c) = d_T(z, c)$. The shortcut pq is not useful for T, because it is indifferent for $\{x, y\}$.

In Figure 3.3, the intersection of the diametral paths of T consists of a single vertex. We argue that a tree with this property cannot have a useful shortcut.

Lemma 3.1. *If the backbone* $\mathcal{B}$ *of a geometric tree* T *consists only of the absolute center* c *of* T*, then there does not exists a useful shortcut for* T*, i.e., every shortcut for* T *is either indifferent or useless.*

Proof. Suppose T is a geometric tree whose backbone $\mathcal{B}$ consists only of the absolute center c. Then, c must be a vertex of T, since $\mathcal{B}$ would contain the edge containing c, otherwise. Let x, y be a diametral pair of T. Then we have $\frac{1}{2}\operatorname{diam}(T) = d_T(x, c) = d_T(y, c)$, since c is the midpoint of the path from x to y in T. Since the backbone only consists of c, there must be at least one other leaf z with $\frac{1}{2}\operatorname{diam}(T) = d_T(z, c)$ such that the paths from z to x and from z to y pass through c. This means x, y, and z lie in three different sub-trees T_X, T_Y, and T_Z attached to c.

Assume, for the sake of a contradiction, that there exists a shortcut pq that is useful for T. Then $p \neq q$, since pq would be indifferent for T, otherwise. Hence, we have $p \neq c$ or $q \neq c$, and one of the sub-trees T_X, T_Y, or T_Z contains neither p nor q. Without loss of generality, suppose $p, q \notin T_X$. By our assumption, pq must be useful for every diametral pair of T including $\{x, y\}$. Without loss of generality, let pq be useful for x, y, i.e., $d_{T+pq}(x, y) = d_T(x, p) + |pq| + d_T(q, y) < d_T(x, y)$. Otherwise, we swap p and q. This implies that q lies in T_Y, since otherwise the shortest path in $T + pq$ from x to y would contain c twice: once along the sub-path from x to p and once along the sub-path from q to y. Since pq is useful for T by assumption, pq must be useful for the unordered pair $\{x, z\}$ and, thus, pq must be useful for either x, z or z, x. The shortcut pq cannot be useful for the ordered pair x, z, since the path from q to z contains c. Therefore, pq must be useful for z, x. This implies that p lies in T_Z, since the path from x to q in $T + pq$ contains c.

In summary, we conclude that pq is useful for x, y and for z, x with $p \in T_Z$ and $q \in T_Y$. This is impossible: Along the shortest path from x to y in $T + pq$ the sub-path from c to p *does not* contain the shortcut pq, i.e., $d_{T+pq}(c, p) = d_T(c, p)$. On the other hand, along the shortest path from x to z in $T + pq$, the sub-path from c to p *does* contain the supposedly useful shortcut pq, i.e., $d_{T+pq}(c, p) < d_T(c, p)$. Therefore, there does not exist a single useful shortcut for T. □

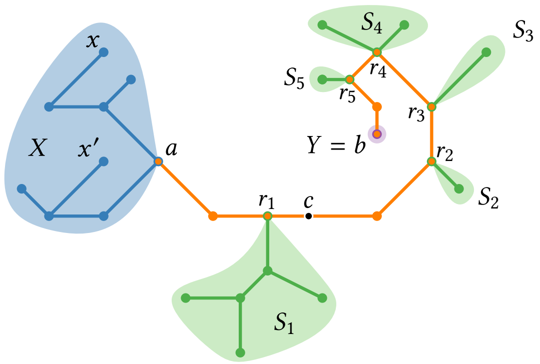


Figure 3.4: The decomposition of the geometric tree T from Figure 3.1 into its backbone $\mathcal{B}$ (orange), its primary $\mathcal{B}$-sub-trees, X (blue) and Y (purple), as well as its secondary $\mathcal{B}$-sub-trees (green). The primary $\mathcal{B}$-sub-tree Y coincides with the endpoint b of $\mathcal{B}$.

Let T be a geometric tree whose backbone $\mathcal{B}$ does not consist of a single vertex, as illustrated in Figure 3.4. In this case, the backbone $\mathcal{B}$ is a path with endpoints a and b such that $a \neq c \neq b$.

We call the sub-trees that are attached to $\mathcal{B}$ the *$\mathcal{B}$-sub-trees* of T. The *root* of a $\mathcal{B}$-sub-tree S is the vertex r connecting S and $\mathcal{B}$. Let X be the $\mathcal{B}$-sub-tree with root a, and let Y be the $\mathcal{B}$-sub-tree with root b. We refer to X and Y as the *primary $\mathcal{B}$-sub-trees*, because every diametral pair of T consists of a leaf x in X and of a leaf y in Y. The other $\mathcal{B}$-sub-trees $S_1, S_2, \ldots, S_k$ with roots r_1, $r_2, \ldots,$ and r_k, respectively, are the *secondary $\mathcal{B}$-sub-trees* of T. Each of the primary $\mathcal{B}$-sub-trees may consist of an endpoint of the backbone only, as exemplified in Figure 3.4.

Theorem 3.2. *For geometric tree T with backbone $\mathcal{B}$, there exist two points $p, q \in T$ such that pq is a useful shortcut for T if and only if $\mathcal{B}$ is neither a line segment nor a single point.*

Proof. Let T be a geometric tree and let pq be a useful shortcut for T. We show that the backbone $\mathcal{B}$ of T is neither a line segment nor a point. Since T possesses a useful shortcut, the backbone $\mathcal{B}$ of T cannot be a point, due to Lemma 3.1. Let a and b be the endpoints of the path $\mathcal{B}$, and let X and Y be the primary $\mathcal{B}$-sub-trees attached to a and b, respectively. Let x, y be a diametral pair of T with $x \in X$ and $y \in Y$. Since pq is useful for T, the shortcut pq must be useful for $\{x, y\}$. Without loss of generality, pq is useful for x, y. Otherwise, we swap p and q.

The shortcut pq is useful for a, b, i.e., $d_T(a,p) + |pq| + d_T(b,q) < d_T(a,b)$, because there exists a diametral pair $\hat{x}, \hat{y}$ of T with $\hat{x} \in X$ and $\hat{y} \in Y$ such that pq is useful for $\hat{x}, \hat{y}$, and a lies on the path from $\hat{x}$ to p, and b lies on the path from q to $\hat{y}$. We locate $\hat{x}$ and $\hat{y}$ as follows.

- Suppose a lies on the path from x to p. Then we let $\hat{x} = x$.
- Suppose a does not lie on the path from x to p. Then $p \in X$ with $p \neq a$. Since a is an endpoint of the intersection of the diametral paths in T, there is some leaf x' of X such that x', y is diametral in T and the path from x' to p contains a. We let $\hat{x} = x'$.

  We argue that pq is useful for x', y. Since pq is useful for T, it must be useful for $\{x', y\}$. The shortcut pq is useful for x, y and, therefore, useful for p, y. Thus, the shortest path connecting p and y in $T + pq$ contains q. If pq was useful for y, x', then the shortest path connecting x' and y in $T + pq$ would travel from x' to q, then via the shortcut qp to p, and then backwards via q to y. This is impossible and, thus, pq must be useful for x', y.

Likewise, we find $\hat{y}$ such that pq is useful for $\hat{x}, \hat{y}$ and b lies on the path from q to $\hat{y}$.

Therefore, pq is useful for a, b. By the triangle inequality, $|ab| \leq d_T(a,p) + |pq| + d_T(q,b)$ and, thus, $|ab| < d_T(a,b)$. This means that the backbone is strictly longer than the line segment connecting its endpoints and, thus, the backbone cannot be a line segment itself.

Conversely, suppose T is a geometric tree whose backbone $\mathcal{B}$ is neither a line segment nor a single point, as illustrated in Figure 3.5. This means that the backbone $\mathcal{B}$ of T is a path with endpoints a and b such that $|ab| < d_T(a,b)$. Let X and Y be the primary $\mathcal{B}$-sub-trees of T attached to a and to b, respectively, and let $S_1, S_2, \ldots, S_k$ be the secondary $\mathcal{B}$-sub-trees of T.

We show that ab is already a useful shortcut for T or we show that we can construct a useful shortcut pq for T with $p, q \in \mathcal{B}$. Let s, t be a diametral pair of $T + ab$. We distinguish three cases: either we have $s, t \in T$ and s, t is diametral in T, or we have $s, t \in T$ and s, t is not diametral in T, or we have $s \notin T$ or $t \notin T$. In the first two cases, we show that ab is useful for T, and in the third case we construct a useful shortcut pq for T with $p, q \in \mathcal{B}$ if ab is not useful for T.

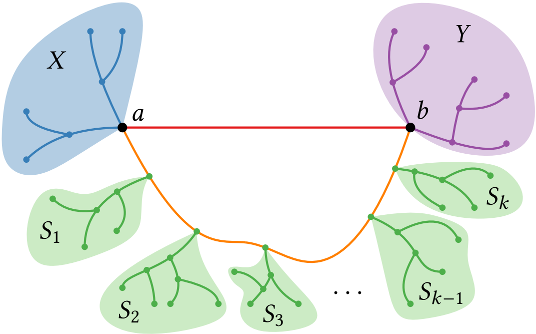


Figure 3.5: A geometric tree whose backbone is not a line segment.

1. Suppose $s, t \in T$ such that s, t is diametral in T, i.e., $d_T(s,t) = \operatorname{diam}(T)$.

   Then s and t lie in different primary $\mathcal{B}$-sub-trees of T. Without loss of generality, we assume that $s \in X$ and $t \in Y$. Otherwise, we swap s and t.

   The shortcut ab is useful for s, t, i.e., $d_T(s,a) + |ab| + d_T(b,y) < d_T(s,t)$, since

   $$d_T(s,a) + |ab| + d_T(b,y) < d_T(s,a) + d_T(a,b) + d_T(b,y) = d_T(s,t) \ .$$

   Therefore, $d_{T+ab}(s,t) < d_T(s,t)$ and, thus, the shortcut ab is useful for T, since

   $$\operatorname{diam}(T+ab) = d_{T+ab}(s,t) < d_T(s,t) = \operatorname{diam}(T) \ .$$

2. Suppose $s, t \in T$ such that s, t is not diametral in T, i.e., $d_T(s,t) < \operatorname{diam}(T)$.

   Then the shortcut ab is useful for T, since

   $$\operatorname{diam}(T+ab) = d_{T+ab}(s,t) \leq d_T(s,t) < \operatorname{diam}(T) \ .$$

3. Suppose $s \notin T$ or $t \notin T$. Without loss of generality, let $s \notin T$. Otherwise, we swap s and t.

   Then $s \in ab$ with $a \neq s \neq b$. Let $C(a,b)$ be the simple cycle in $T + ab$. We distinguish two cases depending on whether t lies on $C(a,b)$ or not.

   a) Suppose $t \in C(a,b)$. Then we have $d_{T+ab}(s,t) < d_T(a,b)$, because

      $$d_{T+ab}(s,t) \leq \frac{|ab| + d_T(a,b)}{2} < \frac{d_T(a,b) + d_T(a,b)}{2} = d_T(a,b) \ .$$

      Therefore, ab is useful for T, since $\operatorname{diam}(T+ab) = d_{T+ab}(s,t) < d_T(a,b) \leq \operatorname{diam}(T)$.

   b) Suppose $t \notin C(a,b)$, as illustrated in Figure 3.6.

      We argue that t is a leaf of a secondary $\mathcal{B}$-sub-tree. Let $\bar{a}$ be the point on $C(a,b)$ that is the farthest point from a with respect to $T + ab$. The point $\bar{a}$ lies in T, because

      $$|ab| = \frac{|ab| + |ab|}{2} < \frac{|ab| + d_T(a,b)}{2} = d_{T+ab}(a,\bar{a}) \ .$$

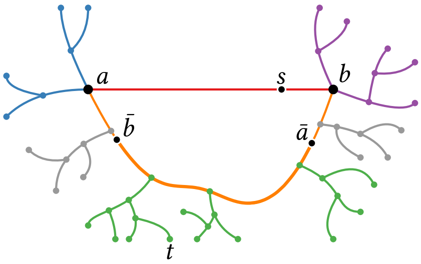


Figure 3.6: A sketch of a geometric tree T whose backbone is not a line segment connecting its endpoints a and b. A diametral pair s, t of $T + ab$ with $s \notin T$. The diametral partner t lies in a $\mathcal{B}$-sub-tree attached to the path from $\bar{a}$ to $\bar{b}$ in T, where $\bar{a}$ and $\bar{b}$ are the respective farthest points from a and from b along the simple cycle $C(a, b)$ in $T + ab$.

Therefore, t cannot lie in X, because $\bar{a}$ is the farthest point on $C(a, b)$ from any point in X, and $s \neq \bar{a}$, since $s \notin T$ and $\bar{a} \in T$. Likewise, we argue that $t \notin Y$. In summary, s lies on the shortcut ab and t is a leaf of a secondary $\mathcal{B}$-sub-tree.

Let δ be the largest diameter of the secondary $\mathcal{B}$-sub-trees $S_1, \ldots, S_k$, i.e.,

$$\delta := \max_{i=1}^{k} \operatorname{diam}(S_i) \ ,$$

and let $\epsilon = \operatorname{diam}(T) - \delta$. We have $\epsilon > 0$, as none of the secondary $\mathcal{B}$-sub-trees contains diametral pairs of T, i.e., $\operatorname{diam}(S_i) < \operatorname{diam}(T)$ for all $i = 1, 2, \ldots, k$. Since $|ab| < d_T(a, b)$, there exist $p, q \in \mathcal{B}$ with $|pq| < d_T(p, q)$ and $|pq| + d_T(p, q) < 2\epsilon$.

We argue that pq is useful for T. Let u, v be a diametral pair of $T + pq$ and let $C(p, q)$ be the simple cycle in $T + pq$. Analogous to diametral pairs of $T + ab$, we argue that pq is useful for T when $u, v \in T$ (Case 1 and 2) and when $u \in pq$ with $p \neq u \neq q$ and $v \in C(p, q)$ (Case 3a). It remains to show that pq is useful for T when $u \in pq$ with $p \neq u \neq q$ and when v is a leaf of a secondary $\mathcal{B}$-sub-tree that is attached to an interior vertex r of the path from p to q in T (Case 3b), as illustrated in Figure 3.7.

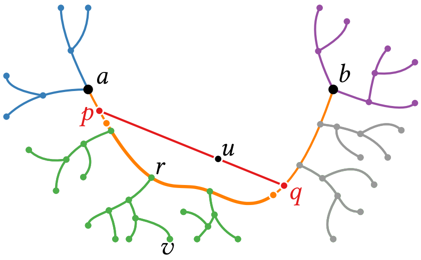


Figure 3.7: A sketch of a geometric tree whose backbone is not a line segment connecting its endpoints a and b. We moved the shortcut to a position pq, since the shortcut ab increased the diameter. The points p and q are chosen such that the simple cycle $C(p, q)$ in $T + pq$ is too small to contain the depicted diametral pair u, v with $u \notin T$.

By definition, we have $d_T(v, r) \leq \delta$. By choice of p and q we have

$$d_{T+pq}(r, u) \leq \frac{|C(p, q)|}{2} = \frac{|pq| + d_T(p, q)}{2} < \frac{2 \cdot \epsilon}{2} = \epsilon \ .$$

Therefore, the shortcut pq is useful for T, since

$$\begin{aligned}
\operatorname{diam}(T + pq) &= d_{T+pq}(u, v) && (u, v \text{ is diametral in } T + pq)\\
&= d_T(v, r) + d_{T+pq}(r, u) && (r \text{ is on any path from } v \text{ to } u \text{ in } T + pq)\\
&< d_T(v, r) + \epsilon && (d_{T+pq}(r, u) < \epsilon)\\
&\leq \delta + \epsilon && (d_T(v, r) \leq \delta)\\
&= \delta + \operatorname{diam}(T) - \delta && (\epsilon = \operatorname{diam}(T) - \delta)\\
&= \operatorname{diam}(T) \ .
\end{aligned}$$

Every diametral pair s, t of $T + ab$ falls into one of the above cases and, in each case, we either argued that ab is useful for T or found a useful shortcut for T when ab was not useful for T. Therefore, T possesses a useful shortcut when $\mathcal{B}$ is not a line segment. □

3.2 Optimal Shortcuts

Consider a geometric tree T whose backbone is a path from a to b. This path contains the absolute center c. We prove that there is an optimal shortcut pq for T such that p lies on the path from a to c and q lies on the path from c to b. This holds when $\mathcal{B}$ consists only of c, i.e., $a = c = b$, since then T has no useful shortcuts and the shortcut cc is indifferent and therefore optimal for T. For the other cases, we establish our claim by proving the following statements.

1. The endpoints of any shortcut pq that is useful for T cannot lie in the same $\mathcal{B}$-sub-tree S.
2. If an endpoint p of a useful, optimal shortcut pq for T lies in a $\mathcal{B}$-sub-tree S with root r, then the shortcut rq is also optimal for T—regardless of the position of q.
3. There exists an optimal shortcut pq for T with $p, q \in \mathcal{B}$.
4. If pq is an optimal shortcut for T with $p, q \in \mathcal{B}$ such that the path from p to q along $\mathcal{B}$ does not contain c, then at least one of pc or cq is also an optimal shortcut for T.

The last statement implies our claim, since c lies on the paths from c to b from c to a. In the following, we prove the above Statements 1 through 4 in Lemmas 3.4 to 3.6 and Theorem 3.7.

The idea for restricting the search for an optimal shortcut along the backbone stems from Große et al. [30] who establish the following theorem for the discrete setting. We generalize their result to the continuous setting and expand it by incorporating the absolute center.

Theorem 3.3 (Discrete Backbone Theorem by Große et al. [30]). *Let T be a geometric tree with vertex set V and backbone $\mathcal{B}$. There exists a pair of vertices $p, q \in \mathcal{B} \cap V$ such that pq minimizes the discrete diameter of the augmented tree $T + pq$ among all possible discrete shortcuts for T, i.e.,*

$$\max_{u,v \in V} d_{T+pq}(u, v) = \min_{r,s \in V} \max_{u,v \in V} d_{T+rs}(u, v) \ .$$

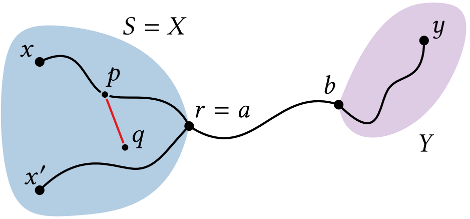


Figure 3.8: A shortcut pq for a geometric tree T that has both endpoints in the same $\mathcal{B}$-sub-tree S with root r and where r does not lie on the path from x to p, which implies $S = X$.

Lemma 3.4. *Let T be a geometric tree, let pq be a shortcut for T, and let S be a $\mathcal{B}$-sub-tree of T. If the shortcut pq is useful for T, then $p \notin S$ or $q \notin S$.*

Proof. Assume, for the sake of a contradiction, that there exists some geometric tree T, with a $\mathcal{B}$-sub-tree S with root r, such that there is a shortcut pq for T with $p, q \in S$ that is useful for T.

Suppose the backbone $\mathcal{B}$ of T is a path that connects the vertices a and b. Let X and Y be the primary $\mathcal{B}$-sub-trees with roots a and b, respectively. Furthermore, let x, y be a diametral pair of T with $x \in X$ and $y \in Y$. Since pq is useful for T, we have $p \neq q$ and pq is useful for x, y or for y, x. Without loss of generality, let pq be useful for x, y. Otherwise, we swap x, y and X, Y.

We distinguish two cases depending on whether the root r of S lies on the path from x to p in T or not. We derive a contradiction by showing that both of these cases are impossible.

1. Suppose r *does not* lie on the path from x to p. Then x, p, and q lie in the same $\mathcal{B}$-sub-tree meaning $S = X$ and $r = a$, since $x \in X$, as illustrated in Figure 3.8. Since y lies in the other primary $\mathcal{B}$-sub-tree, we have $y \notin S$ and, thus, r lies on the path from q to y.

   Since a is an endpoint of the backbone, i.e., the intersection of all diametral paths in T, there is a leaf x' of X such that x', y is diametral in T and r is on the path from x' to p.

   The shortcut pq cannot be useful for x', y, since r lies on the path from x' to p and on the path from q to y. Thus, for pq to be useful for T, the shortcut pq must be useful for y, x'. This means that the shortest path from y to x' in $T + pq$ contains the path from r to p in T, i.e., $d_T(r, p) < d_T(r, q) + |qp|$. On the other hand, the shortest path from y to x in $T + pq$ travels from r via the shortcut to p, i.e., $d_T(r, q) + |qp| < d_T(r, p)$. This is impossible.

2. Suppose r lies on the path from x to p. Then r *does not* lie on the path from y to q and, thus, y, q, and p lie in the same $\mathcal{B}$-sub-tree, meaning $S = Y$ and $r = b$ and $x \notin S$. This situation is symmetric to the previous case and, therefore, impossible as well.

Therefore, if the shortcut pq is useful for T, then $p \notin S$ or $q \notin S$. □

Lemma 3.5. *Let T be a geometric tree and let S be a $\mathcal{B}$-sub-tree of T with root r. If pq is a useful shortcut for T with $p \in S$ then* $\operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$.

Remark. The proof of Lemma 3.5 below follows the proof of the discrete backbone lemma by Große et al. [30]. The first two cases are due to Große et al. [30]; they are provided for the sake of self-containment. We add the third case to generalize the result to the continuous setting.

Proof of Lemma 3.5. Let T be a geometric tree, let S be a $\mathcal{B}$-sub-tree of T with root r, and let pq be a useful shortcut for T with $p \in S$. Finally, let s, t be a diametral pair of $T + rq$.

We distinguish three cases: Either we have $s, t \in T$ and pq is indifferent for $\{s, t\}$, or we have $s, t \in T$ and pq is useful for $\{s, t\}$, or we have $s \notin T$ or $t \notin T$. In each case, we argue that $d_{T+rq}(s, t) \leq d_{T+pq}(s', t')$ for some $s', t' \in T + pq$ and, thus, $\operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$.

1. Suppose $s, t \in T$ and pq is indifferent for $\{s, t\}$, i.e., $d_T(s, t) = d_{T+pq}(s, t)$.

   Then we have $\operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$, because

$$\operatorname{diam}(T + rq) = d_{T+rq}(s, t) \leq d_T(s, t) = d_{T+pq}(s, t) \leq \operatorname{diam}(T + pq) \ .$$

2. Suppose $s, t \in T$ and pq is useful for $\{s, t\}$, i.e., $d_T(s, t) > d_{T+pq}(s, t)$.

   Then pq is useful for s, t or t, s. We assume, without loss of generality, that pq is useful for s, t, i.e., $d_T(s, p) + |pq| + d_T(q, t) = d_{T+pq}(s, t) < d_T(s, t)$. Otherwise, we swap s and t.

   We distinguish two cases depending on whether the path from s to p in T contains r.

   a) Suppose the path from s to p in T contains r, as shown in Figure 3.9.

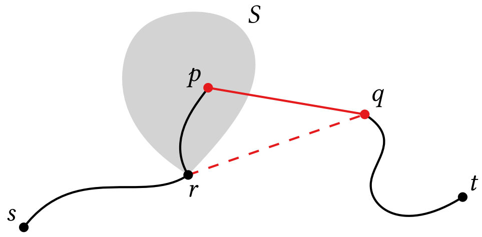


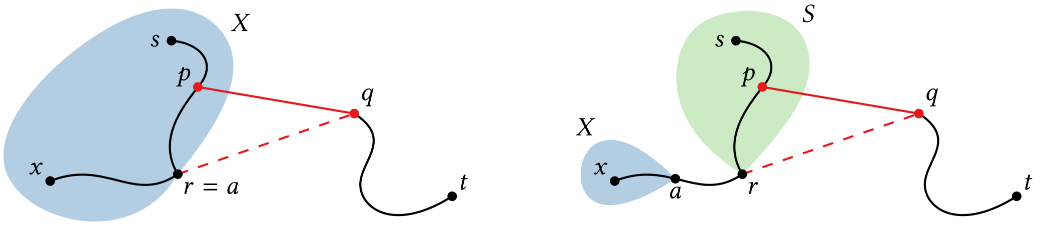
Figure 3.9: A sketch of the shortest path in $T + pq$ connecting s and t via pq where p lies in some $\mathcal{B}$-sub-tree S whose root r lies on the path from s to p in T. Regardless of whether $s \in S$ or $s \notin S$, we have $q \notin S$ by Lemma 3.4, and, therefore $t \notin S$.

   Then we have $\operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$, because

$$\begin{aligned}
\operatorname{diam}(T + rq) &= d_{T+rq}(s, t) && (s, t \text{ is diametral in } T + rq)\\
&\leq d_T(s, r) + |rq| + d_T(q, t) && (\text{triangle inequality for } d_{T+rq})\\
&\leq d_T(s, r) + d_T(r, p) + |pq| + d_T(q, t) && (\text{triangle inequality for } |.|)\\
&= d_T(s, p) + |pq| + d_T(q, t) && (\text{the path from } s \text{ to } p \text{ in } T \text{ contains } r)\\
&= d_{T+pq}(s, t) && (pq \text{ is useful for } s, t)\\
&\leq \operatorname{diam}(T + pq) \ .
\end{aligned}$$

   b) Suppose the path from s to p in T *does not* contain r, as shown in Figure 3.10.

   Let a and b be the endpoints of the backbone $\mathcal{B}$ of T, and let X and Y be the primary $\mathcal{B}$-sub-trees with roots a and b, respectively. We argue that there exists a diametral pair x, y of T with $x \in X$ and $y \in Y$ such that r lies on the path from x to p.



(a) The case $S = X$.

(b) The case $S \neq X$.

Figure 3.10: A sketch of the shortest path in $T + pq$ connecting s and t via pq where p lies in some $\mathcal{B}$-sub-tree S and p lies on the path from s to the root r of S. There is a diametral partner x of T such that the path from x to p passes through r. This property holds regardless of whether S is a primary $\mathcal{B}$-sub-tree (a) or a secondary $\mathcal{B}$-sub-tree (b).

Every path in $T + pq$ that connects a diametral pair of T contains the shortcut pq, since pq is useful for T. If $S = X$, then there is a diametral pair x, y of T with $x \in X$ and $y \in Y$ such that the path from x to p contains $r = a$, as illustrated in Figure 3.10a. Otherwise, a would not be the endpoint of the backbone. If $S \neq X$, then r lies on the path from x to p for every $x \in X$, since $p \in S$, as illustrated in Figure 3.10b.

The shortcut pq is useful for x, y and, thus, for r, q, since r lies on the path from x to p. Therefore, no shortest path in $T + pq$ may contain the path connecting r and q. This includes the path from s to t. Hence, $t \notin S$ and $d_{T+rq}(s,t) = d_T(s,r) + d_{T+rq}(r,t)$.

By Lemma 3.4, $p \in S$ implies $q \notin S$. Since the path from x to y in $T + pq$ cannot contain r two times, we have $y \notin S$ and the path in $T + rq$ from x to t contains r, i.e., $d_{T+rq}(x,t) = d_T(x,r) + d_{T+rq}(r,t)$. We have $d_T(s,r) \leq d_T(x,r)$, because x, y is a diametral path of T and r lies on the path from x to y and on the path from s to y. This implies that x, t is a diametral pair of $T + rq$, because

$$\begin{aligned}
\operatorname{diam}(T + rq) &= d_{T+rq}(s,t) && (s, t \text{ is diametral in } T + rq)\\
&= d_T(s,r) + d_{T+rq}(r,t) && (s \in S \text{ and } t \notin S)\\
&\leq d_T(x,r) + d_{T+rq}(r,t) && (d_T(s,r) \leq d_T(x,r))\\
&= d_{T+rq}(x,t) && (\text{any path from } x \text{ to } t \text{ in } T + rq \text{ contains } r)\\
&\leq \operatorname{diam}(T + rq)\ .
\end{aligned}$$

More precisely, x, t is a diametral pair of $T + rq$ such that the path from x to p contains r. With the argument from the previous case, we obtain

$$\operatorname{diam}(T + rq) = d_{T+rq}(s,t) = d_{T+rq}(x,t) \leq \operatorname{diam}(T + pq)\ .$$

3. Suppose $s \notin T$ or $t \notin T$. Without loss of generality, let $s \notin T$. Otherwise, we swap s and t.

   Then we have $s \in rq$ with $r \neq s \neq q$, as illustrated in Figure 3.11.

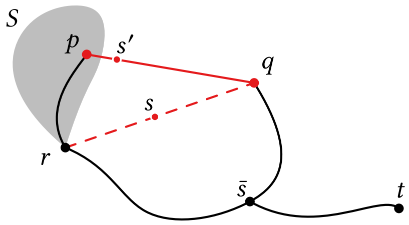


Figure 3.11: A diametral pair s, t of $T + rq$ with $s \in rq$. We move the endpoint of the shortcut at $r \in \mathcal{B}$ to a point p in a $\mathcal{B}$-sub-tree S such that pq is useful for T.

Let $C(r, q)$ be the simple cycle in $T + rq$. The path from s to t leaves $C(r, q)$ at the point $\bar{s} \in C(r, q)$ with $d_{T+rq}(s, \bar{s}) = \frac{1}{2}(d_T(r, q) + |rq|)$. Note that $r \neq \bar{s} \neq q$, because $r \neq s \neq q$ and $|rq| \leq d_T(r, q)$. By Lemma 3.4, $p \in S$ implies $q \notin S$. This means the path connecting r and q in T lies outside of S and, therefore, $\bar{s} \notin S$ and $t \notin S$.

The cycle $C(p, q)$ is formed by pq and the path from p to q in T. Since r lies on the path from p to q, we know that $\bar{s} \in C(p, q)$. Let s' be the farthest point from $\bar{s}$ on $C(p, q)$.

Then we have $\operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$, because

$$\begin{aligned}
\operatorname{diam}(T + rq) &= d_{T+rq}(s, t) && (s, t \text{ is diametral in } T + rq)\\
&= d_{T+rq}(s, \bar{s}) + d_T(\bar{s}, t) && (\bar{s} \text{ lies on any path from } t \text{ to } s \text{ in } T + rq)\\
&= \frac{d_T(r, q) + |rq|}{2} + d_T(\bar{s}, t) && (s \text{ and } \bar{s} \text{ are antipodals on } C(r, q))\\
&\leq \frac{d_T(r, q) + d_T(r, p) + |pq|}{2} + d_T(\bar{s}, t) && (\text{triangle inequality})\\
&= \frac{d_T(p, q) + |pq|}{2} + d_T(\bar{s}, t) && (r \text{ lies on the path from } p \text{ to } q \text{ in } T)\\
&= d_{T+pq}(s', \bar{s}) + d_T(\bar{s}, t) && (s' \text{ and } \bar{s} \text{ are antipodals on } C(p, q))\\
&= d_{T+pq}(s', t) && (\bar{s} \text{ lies on any path from } t \text{ to } s' \text{ in } T + pq)\\
&\leq \operatorname{diam}(T + pq) \ .
\end{aligned}$$

Therefore, if pq is a useful shortcut for T with $p \in S$, then $\operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$. □

Lemma 3.6 (Continuous Backbone Lemma). *Let T be a geometric tree T with backbone $\mathcal{B}$. There exist $p, q \in \mathcal{B}$ such that pq is an optimal shortcut for T.*

Proof. Let pq be an optimal shortcut for a geometric tree T. We assume that pq is useful for T, since otherwise the shortcut cc is indifferent for T with $c \in \mathcal{B}$ and, thus, satisfies the claim.

If $p, q \in \mathcal{B}$, we are done. Suppose $p \notin \mathcal{B}$, i.e., p lies in some $\mathcal{B}$-sub-tree S with root $r \in \mathcal{B}$. According to Lemma 3.5 this implies $\operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$. This means that rq is also an optimal shortcut for T. If $q \in \mathcal{B}$, then the claim follows. Otherwise, $q \notin \mathcal{B}$, i.e., q lies in some $\mathcal{B}$-sub-tree S' with root $r' \in \mathcal{B}$. Since pq is useful for T and $p \in S$, Lemma 3.4 implies that $q \notin S$ and, thus, $S \neq S'$ and $r \neq r'$. By Lemma 3.5, this implies $\operatorname{diam}(T + rr') \leq \operatorname{diam}(T + rq) \leq \operatorname{diam}(T + pq)$, and, thus, rr' is an optimal shortcut for T with both endpoints along the backbone of T. □

Theorem 3.7. *Let T be a geometric tree with backbone $\mathcal{B}$ and absolute center c. There exist $p, q \in \mathcal{B}$ such that pq is an optimal shortcut for T and such that c lies on the path from p to q in T.*

Proof. Let T be a geometric tree with backbone $\mathcal{B}$ and absolute center c. By Lemma 3.6, there exists an optimal shortcut pq for T with $p, q \in \mathcal{B}$. If pq is indifferent for T, then the degenerate shortcut cc satisfies the claim. Thus, we assume that pq is useful for T.

Since there exists a useful shortcut for T, the backbone $\mathcal{B}$ of T is a path with endpoints a and b that contains c with $a \neq c \neq b$. Suppose c does not lie on the path from p to q along $\mathcal{B}$. Without loss of generality, we assume that p and q lie on the path from a to c with $p \neq c \neq q$. Otherwise, we swap a and b. Furthermore, we assume, without loss of generality, that the path from p to c along $\mathcal{B}$ contains q, i.e., $d_T(q, c) \leq d_T(p, c)$. Otherwise, we swap p and q.

We argue that the shortcut pc is at least as good as pq, i.e., $\operatorname{diam}(T + pc) \leq \operatorname{diam}(T + pq)$. Let s, t be a diametral pair of $T + pc$. Either we have $s, t \in T$ and pq is indifferent for $\{s, t\}$, or we have $s, t \in T$ and pq is useful for $\{s, t\}$, or we have $s \notin T$ or $t \notin T$. We show $d_{T+pc}(s, t) \leq d_{T+pq}(s, t)$ and, thus, $\operatorname{diam}(T + pc) \leq \operatorname{diam}(T + pq)$, for the first two cases and we rule out the third case.

1. Suppose $s, t \in T$ and pq is indifferent for $\{s, t\}$, i.e., $d_T(s, t) = d_{T+pq}(s, t)$.

   Then we have $\operatorname{diam}(T + pc) \leq \operatorname{diam}(T + pq)$, because

   $$\operatorname{diam}(T + pc) = d_{T+pc}(s, t) \leq d_T(s, t) = d_{T+pq}(s, t) \leq \operatorname{diam}(T + pq) \enspace .$$

2. Suppose $s, t \in T$ and pq is useful for $\{s, t\}$, i.e., $d_{T+pq}(s, t) < d_T(s, t)$.

   Then pq is useful for s, t or t, s. Without loss of generality, let pq be useful for s, t. Otherwise, we swap s and t. Figure 3.12 illustrates the following arguments.

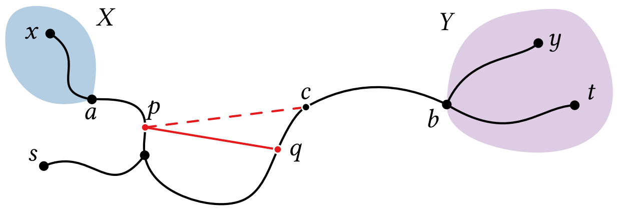


Figure 3.12: A sketch showing a shortcut pq for a tree T with both endpoints on one side of the absolute center c together with a diametral pair s, t of $T + pc$ for which pq is useful.

   Let X and Y be the primary $\mathcal{B}$-sub-trees with roots a and b, respectively, and let x, y be a diametral pair of T with $x \in X$ and $y \in Y$. The shortcut pq is useful for T and, thus, pq is useful for $\{x, y\}$. Hence, pq is useful for x, y, since p lies on the path from x to q in T.

   The path from s to p cannot contain q, since pq is useful for s, t. This means s lies in the largest sub-tree of T with leaves x and q and the path from s to c in T contains q. In the following, we argue $t \in Y$, which implies that c lies on the path from s to t in $T + pc$.

   As pq is useful for s, t, the shortcut pq is useful for s, q, i.e., $d_T(s, p) + |pq| < d_T(s, q)$, since

   $$d_T(s, p) + |pq| + d_T(q, t) = d_{T+pq}(s, t) < d_T(s, t) \leq d_T(s, q) + d_T(q, t) \enspace .$$

This means pc is also useful for s, c, i.e., $d_T(s,p) + |pc| < d_T(s,c)$, since

$$d_T(s,p) + |pc| \leq d_T(s,p) + |pq| + d_T(q,c) < d_T(s,q) + d_T(q,c) = d_T(s,c) \ .$$

The point t is a farthest leaf from c in T, i.e., $d_T(c,t) = d_T(c,y)$, because

$$\begin{aligned} d_T(s,p) + |pc| + d_T(c,t) &\leq d_T(s,p) + |pc| + d_T(c,y) && (y \text{ is a farthest leaf from } c \text{ in } T) \\ &= d_{T+pc}(s,c) + d_T(c,y) && (pc \text{ is useful for } s,c) \\ &= d_{T+pc}(s,y) && (c \text{ lies on the path from } s \text{ to } y) \\ &\leq d_{T+pc}(s,t) && (s,t \text{ is diametral in } T+pc) \\ &\leq d_T(s,p) + |pc| + d_T(c,t) \ . \end{aligned}$$

Since pq was useful for s, t, the path from q to t cannot contain p. On the other hand, p lies on the backbone and blocks the path from q to any farthest leaf from c in X. Since t is a farthest leaf from c in T, this means that $t \in Y$ and the claim follows, since

$$\begin{aligned} \operatorname{diam}(T+pc) &= d_{T+pc}(s,t) && (s,t \text{ diametral in } T+pc) \\ &\leq d_T(s,p) + |pc| + d_T(c,t) \\ &\leq d_T(s,p) + |pq| + d_T(q,c) + d_T(c,t) && (\text{triangle inequality}) \\ &= d_T(s,p) + |pq| + d_T(q,t) && (c \text{ is on the path from } q \text{ to } t) \\ &= d_{T+pq}(s,t) && (pq \text{ is useful for } s,t) \\ &\leq \operatorname{diam}(T+pq) \ . \end{aligned}$$

3. Suppose $s \notin T$ or $t \notin T$. Without loss of generality, $s \notin T$, i.e., $s \in pc$ with $p \neq s \neq c$.

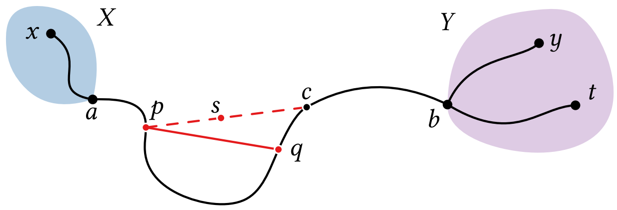


Figure 3.13: A sketch of a shortcut pq for a tree T with both endpoints on one side of the absolute center c together with an impossible diametral pair s, t of $T + pc$ with $s \notin T$.

Let X and Y be the primary $\mathcal{B}$-sub-trees with roots a and b, respectively. As illustrated in Figure 3.13, the point s lies on the simple cycle $C(p,c)$ in $T + pc$. The largest tree attached to $C(p,c)$ in $T + pc$ is the one containing Y, since c is the absolute center of T and both p and q lie on the path from a to c. Therefore, the point t lies in Y and s is the farthest point from t on $C(p,c)$. The path from t to s in $T + pc$ enters $C(p,c)$ at c. Therefore, s is the farthest point from c on $C(p,c)$. However, this implies that s lies on the path from p to c in T, since $|pc| \leq d_T(p,c)$ contradicting $s \notin T$. This means that this case is impossible.

We conclude that if pq is an optimal shortcut for T such that p and q lie on the path from a to c in T with $d_T(q,c) \leq d_T(p,c)$ then pc is also an optimal shortcut for T. Therefore, there exist $p, q \in \mathcal{B}$ such that pq is an optimal shortcut for T and the path from p to q in T contains c. ☐

3.3 Preparations for the Algorithm

Our search for an optimal shortcut pq for T proceeds as follows. Initially, we place the endpoints of the shortcut, p and q, at the absolute center c of T. Then, we move p and q along the backbone $\mathcal{B}$ balancing the diametral paths in $T + pq$. Throughout this movement p remains along the path from a to c and q remains on the path from c to b, where a and b are the endpoints of $\mathcal{B}$.

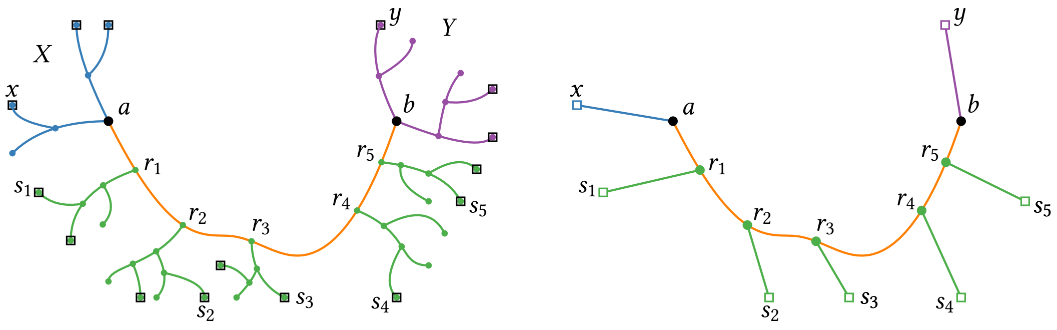
The diametral pairs in $T + pq$ guide our search: each diametral pair in $T + pq$ rules out some direction in which we could search for a better shortcut. We have found an optimal shortcut when p and q reach a position where the diametral pairs block all directions of movement, except perhaps going back the way we came. We describe our algorithm along the following steps.

1. We simplify the geometric tree T by compressing the $\mathcal{B}$-sub-trees, thereby simplifying the discussion about diametral pairs and paths in the augmented tree $T + pq$.
2. We define algorithm states in terms of the diametral paths and diametral pairs that are present in the augmented tree, and we distinguish four types of movements for the shortcut—called *in-shift*, *out-shift*, x*-shift*, and y*-shift*—as the operations of our algorithm.
3. We observe how each type of diametral pair rules out a better shortcut in some direction, and that some combinations of these types imply that the current shortcut is optimal.
4. We describe the continuous, conceptual movement of the shortcut that is guided by the set of types of diametral pairs that are present in $T + pq$. We identify the invariants that are upheld by this movement and that guarantee that we find an optimal shortcut.
5. We specify the speeds at which the endpoints of the shortcut would move in the continuous algorithm. These speeds depend on the set of types of diametral paths in $T + pq$; the changes in this set constitute the events for the discretization.
6. We bound the number of events that the discrete algorithm needs to process by $O(n)$. This involves ruling out some transitions between the algorithm states as well as identifying situations were we can safely ignore events without compromising optimality.
7. Finally, we explain how we can process each of the $O(n)$ events in $O(\log n)$ amortized time and, thus, bound the running time of our algorithm by $O(n \log n)$.

In this section, we discuss Steps 1, 2, and 3, i.e., the preparations for the algorithm. In Section 3.4, we describe Step 4, i.e., the continuous algorithm and its correctness. In Section 3.5, we discuss Steps 5 through 7, i.e., the discretization of the algorithm and its running time.

3.3.1 Simplifying the Tree

Let T be a geometric tree whose backbone B consists of more than its absolute center c. Let a and b be the endpoints of $\mathcal{B}$, and let X and Y be the primary $\mathcal{B}$-sub-trees of T with roots a and b, respectively, and let $S_1, S_2, \ldots, S_k$ be the secondary $\mathcal{B}$-sub-trees of T that are attached to $\mathcal{B}$ at their roots $r_1, r_2, \ldots, r_k$, respectively. Let x be a farthest leaf from a in X, let y be a farthest leaf from b in Y and, for every $i = 1, 2, \ldots, k$, let s_i be a farthest leaf from r_i in S_i, as in Figure 3.14a.



(a) A geometric tree T.

(b) The perspective from $\mathcal{B}$ on T.

Figure 3.14: An illustration of a geometric tree (a) together with the perspective from its backbone (b). We represent each $\mathcal{B}$-sub-tree S with an edge whose length is the length of a path from the root of S to a farthest leaf (squares) of S.

We simplify the discussion about diametral pairs in $T+pq$. First, there is no need to distinguish diametral pairs with partners in the same $\mathcal{B}$-sub-trees: for any $i, j = 1, 2, \ldots, k$ with $i \neq j$, the pair s_i, s_j is diametral in $T+pq$ if and only if every farthest leaf from r_i in S_i forms a diametral pair with every farthest leaf from r_j in S_j. Second, there is no need to consider diametral pairs with both endpoints in the same $\mathcal{B}$-sub-tree, as we argue in Lemma 3.8 below. Therefore, we simplify T by replacing each $\mathcal{B}$-sub-tree S_i with an edge from r_i to a vertex representing s_i of length $d(r_i, s_i)$. Likewise, we replace X and Y with edges of appropriate length, as illustrated in Figure 3.14. We refer to the resulting caterpillar network as the *perspective* from $\mathcal{B}$ on T.

Lemma 3.8. *Let T be a geometric tree with backbone $\mathcal{B}$, let $p, q \in \mathcal{B}$, and let S be a $\mathcal{B}$-sub-tree of T. If there exist $u, v \in S$ such that u, v is diametral in $T+pq$, then pq is an optimal shortcut for T.*

Proof. Let T be a geometric tree with backbone $\mathcal{B}$, let $p, q \in \mathcal{B}$, and let S be a $\mathcal{B}$-sub-tree of T. Suppose there exist $u, v \in S$ such that u, v is a diametral pair of $T+pq$.

Since S is a tree that is attached to the remainder of $T+pq$, we have $\operatorname{diam}(S) \leq \operatorname{diam}(T+pq)$. Every path from u to v via pq contains r twice, hence the shortest path from u to v in $T+pq$ remains in S. This implies $\operatorname{diam}(T+pq) = \operatorname{diam}(S) = d_S(u, v)$, since

$$\operatorname{diam}(T+pq) = d_{T+pq}(u, v) = d_S(u, v) \leq \operatorname{diam}(S) \leq \operatorname{diam}(T+pq) \;.$$

By Lemma 3.6, there is an optimal shortcut p^*q^* for T with $p^*, q^* \in \mathcal{B}$. By repeating the above, we obtain $d_S(u, v) = d_{T+p^*q^*}(u, v)$ and, thus, $\operatorname{diam}(T+pq) = \operatorname{diam}(T+p^*q^*)$, since

$$\operatorname{diam}(T+p^*q^*) \leq \operatorname{diam}(T+pq) = \operatorname{diam}(S) = d_S(u, v) = d_{T+p^*q^*}(u, v) \leq \operatorname{diam}(T+p^*q^*) \;.$$

Therefore, pq is an optimal shortcut for T and $\operatorname{diam}(T+pq) = \operatorname{diam}(S)$. ☐

Corollary 3.9. *Let T be a geometric tree, and let δ be the largest diameter of any $\mathcal{B}$-sub-tree of T, i.e., $\delta := \max\{\operatorname{diam}(S) \mid S$ is a $\mathcal{B}$-sub-tree of $T\}$. For every shortcut pq for T, we have $\delta \leq \operatorname{diam}(T+pq)$.*

We compute the largest diameter δ of any $\mathcal{B}$-sub-tree of T as part of our preprocessing and we halt our search for an optimal shortcut if the current diameter reaches δ. This is not strictly necessary: if we ignore diametral pairs in the same $\mathcal{B}$-sub-tree, we still obtain an optimal shortcut even though we might not know the actual value of the optimal diameter. Nevertheless, we may safely exclude diametral pairs in the same $\mathcal{B}$-sub-tree from consideration.

3.3.2 States and Operations

We group the diametral pairs and paths of the augmented tree $T + pq$ into types. The types of diametral pairs and paths in $T + pq$ define the states of our algorithm. We specify four different types of movements for the shortcut as the base operations for the algorithm.

Pair States

Every partner v of a diametral pair u, v in $T + pq$ is either: (x, y) a diametral partner in the tree, (■) some other point on the tree, or (○) a point on the shortcut. More specifically, v is either (x, y) a leaf of a primary $\mathcal{B}$-sub-tree, (▲) a leaf of a secondary $\mathcal{B}$-sub-tree, or v is a point on the simple cycle $C(p, q)$ in the augmented tree that (•) lies in the original tree or (○) on the shortcut. This leads to the following distinction of the diametral pairs in $T + pq$.

- **(x-y)**: Diametral pairs x, y of $T + pq$ with $x \in X$ and $y \in Y$.
- **(x-■)**: Diametral pairs x, v of $T + pq$ with $x \in X$ and $v \in T \setminus (X \cup Y)$.

  Diametral pairs of this type manifest as one of the following two sub-types.

  - **(x-▲)**: Diametral pairs x, s_j of $T + pq$ with $x \in X$ and $s_j \in S_j$ for some $j = 1, 2, \dots, k$.
  - **(x-•)**: Diametral pairs $x, \bar{x}$ of $T + pq$, where $x \in X$ and where $\bar{x}$ is the farthest point from x on $C(p, q)$. Since $|pq| \leq d(p, q)$, we always have $\bar{x} \in T$.
- **(■-y)**: Diametral pairs u, y of $T + pq$ with $u \in T \setminus (X \cup Y)$ and $y \in Y$.

  Diametral pairs of this type manifest as one of the following two sub-types.

  - **(▲-y)**: Diametral pairs s_i, y of $T + pq$ with $s_i \in S_i$ and $y \in Y$ for some $i = 1, 2, \dots, k$.
  - **(•-y)**: Diametral pairs $y, \bar{y}$ of $T + pq$, where $y \in Y$ and where $\bar{y}$ is the farthest point from y on $C(p, q)$. Since $|pq| \leq d(p, q)$, we always have $\bar{y} \in T$.
- **(■-■)**: Diametral pairs u, v of $T + pq$ with $u, v \in T \setminus (X \cup Y)$.

  Diametral pairs of this type manifest as one of the following two sub-types.

  - **(▲-▲)**: Diametral pairs s_i, s_j of $T + pq$ with $s_i \in S_i$ and $s_j \in S_j$ for $i, j = 1, 2, \dots, k$.
  - **(▲-•)**: Diametral pairs $s_i, \bar{s}_i$ of $T + pq$ with $s_i, \bar{s}_i \in T$ where $s_i \in S_i$ for some $i = 1, 2, \dots, k$ and where $\bar{s}_i \in T$ is the farthest point from s_i on $C(p, q)$.
- **(■-○)**: Diametral pairs u, v of $T + pq$ with $v \notin T$, i.e., $v \in pq$ with $p \neq v \neq q$.

  Since $|pq| \leq d(p, q)$, we have $u \in T$. These diametral pairs only manifest as diametral pairs of the form $s_i, \bar{s}_i$ with $s_i \in S_i$, for $i = 1, 2, \dots, k$, and where $\bar{s}_i$ is the farthest point from s_i on $C(p, q)$ that happens to lie in the interior of the shortcut pq.

There is no need to consider diametral pairs of type ●-● or ●-○: the distance from x or y to their respective farthest points on the cycle $C(p,q)$ is always larger than the distance between any two points on $C(p,q)$—unless the augmented tree is a cycle, i.e., $T+pq=C(p,q)$.

There are no diametral pairs of type x-○ or ○-y, as $\bar{x},\bar{y}\in T$. If x,v is a diametral pair of type x-■ in $T+pq$ then $v\in T$ and if u,y is of type ■-y in $T+pq$ then $u\in T$.

The *pair state* is the set of types of diametral pairs in $T+pq$ and the *pair sub-state* is the set of sub-types of diametral pairs that are present in $T+pq$. For instance, if $T+pq$ has the diametral pairs x,y; x,s_3; x,s_5; and $x,\bar{x}$ then $T+pq$ is in pair state {x-y, x-■} and, more precisely, $T+pq$ is in the pair sub-state {x-y, x-▲, x-●}. Another example is shown in Figure 3.15.

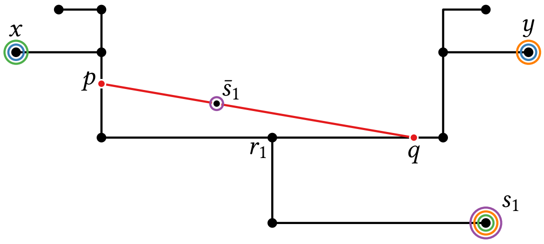


Figure 3.15: An optimal shortcut pq for the geometric tree T from Figure 3.2. The augmented tree $T+pq$ is in pair state {x-y, x-■, ■-y, ■-○} as attested by the diametral pairs x,y (blue); x,s_1 (green); s_1,y (orange); and $s_1,\bar{s}_1$ (purple). Since pq is useful for x,y, but useless for x,s_1 and s_1,y, the path state of $T+pq$ is {x-pq-y, x-T-▲, ▲-T-y, ▲-p-○, ▲-q-○}.

Path States

Let u,v be a diametral pair in $T+pq$. If $u,v\in T$ then every diametral path in $T+pq$ that connects u and v either contains the shortcut (∗-pq-∗) or not (∗-T-∗). If $u\in T$ and $v\notin T$, then every diametral path in $T+pq$ that connects u and v contains either p (∗-p-∗) or q (∗-q-∗). There are no diametral pairs with $u,v\notin T$. This leads to the following distinction.

- **(∗-*pq*-∗)**: A diametral path that *does* contain pq and connects $u\in T$ with $v\in T$.
- **(∗-*T*-∗)**: A diametral path that *does not* contain pq and connects $u\in T$ with $v\in T$.
- **(∗-*p*-∗)**: A diametral path that contains p and connects $u\in T$ with $v\notin T$.
- **(∗-*q*-∗)**: A diametral path that contains q and connects $u\in T$ with $v\notin T$.

In this notation, any type of diametral partner (e.g., x, y, ■, ▲, ●, ○) may appear in place of ∗. For instance, we denote a diametral path from x to $\bar{x}$ via the shortcut by x-pq-●.

The *path state* is the set of types of diametral paths that are present in $T+pq$. For instance, if $T+pq$ has the diametral pairs x,y; x,s_3; x,s_5; and $x,\bar{x}$ such that pq is useful for x,y and x,s_3 but useless for x,s_5 then $T+pq$ is in path state {x-pq-y, x-pq-▲, x-T-▲, x-pq-●, x-T-●}.

Operations

We distinguish the four types of movements for the shortcut that are illustrated in Figure 3.16. Suppose we move a shortcut pq for T with $p, q \in \mathcal{B}$ such that $d(a,p) \leq d(a,q)$ to some new position $p'q'$ with $p', q' \in \mathcal{B}$ such that $d(a,p') \leq d(a,q')$. The movement from pq to $p'q'$ is

- an *outward shift* when p moves towards a and q moves towards b, i.e., when $d(a,p') \leq d(a,p)$ and $d(b,q') \leq d(b,q)$ and, thus $d(a,p') \leq d(a,p) \leq d(a,q) \leq d(a,q')$,
- an *inward shift* when p moves away from a and q moves away from b, i.e., when $d(a,p) \leq d(a,p')$ and $d(b,q) \leq d(b,q')$ and, thus, $d(a,p) \leq d(a,p') \leq d(a,q') \leq d(a,q)$,
- a *shift towards* x when p moves towards a and q moves away from b, i.e., when $d(a,p') \leq d(a,p)$ and $d(b,q) \leq d(b,q')$ and, thus, $d(a,p') \leq d(a,p) \leq d(a,q') \leq d(a,q)$, or
- a *shift towards* y when p moves away from a and q moves towards b, i.e., when $d(a,p) \leq d(a,p')$ and $d(b,q') \leq d(b,q)$ and, thus, $d(a,p) \leq d(a,p') \leq d(a,q) \leq d(a,q')$.

These types of movements intentionally overlap when one of the endpoints remains stationary, e.g., when $p = p'$ every shift towards y is also an outwards shift, as illustrated in Figure 3.16e.

The operations, as defined above, allow us to move p or q through the absolute center c. However, this never happens: the algorithm automatically maintains that p stays on the path from a to c and q stays on the path from c to b—without taking any special care to ensure this.

3.3.3 Blocking

Each type of diametral pair serves as a witness that some type of movement cannot lead to a better shortcut, i.e., cannot reduce the continuous diameter. For instance, if an augmented tree $T + pq$ has a diametral pair of type x-y, then the distance between x and y will increase or remain the same when we shift pq inward. In this sense, x-y *blocks* any inward shift, x-■ *blocks* any y-shift, ■-y *blocks* any x-shift, and ■-■ or ■-○ *block* any outward shift.

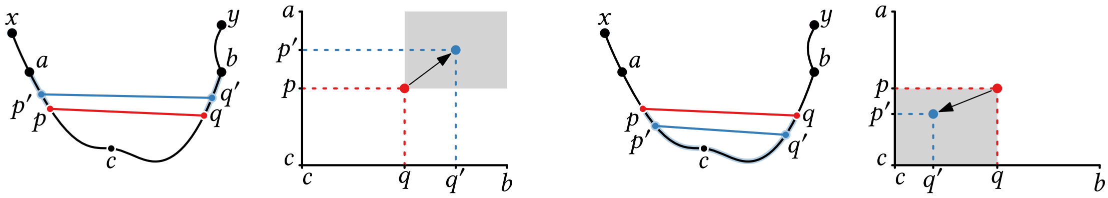
Lemma 3.10 (Blocking Lemma). *Let pq be a shortcut for a tree T with $p, q \in \mathcal{B}$.*

(1) If $T + pq$ has a diametral pair of type x-y, then $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$ *for every shortcut $p'q'$ such that the movement from pq to $p'q'$ is an inward shift.*

(2) If $T + pq$ has a diametral pair of type x-■, then $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$ *for every shortcut $p'q'$ such that the movement from pq to $p'q'$ is a shift towards y.*

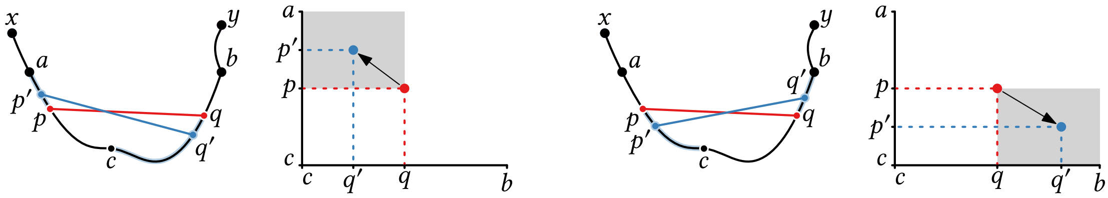
(3) If $T + pq$ has a diametral pair of type ■-y, then $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$ *for every shortcut $p'q'$ such that the movement from pq to $p'q'$ is a shift towards x.*

(4) If $T + pq$ has a diametral pair of type ■-■ or ■-○, then $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$ *for every shortcut $p'q'$ such that the movement from pq to $p'q'$ is an outward shift.*



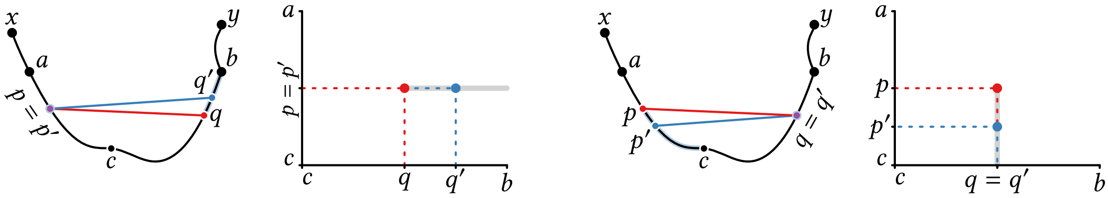
(a) An outwards shift.

(b) An inwards shift.



(c) A shift towards x.

(d) A shift towards y.



(e) An outwards shift towards y.

(f) An inwards shift towards y.

Figure 3.16: The four ways to move a shortcut pq to a new position $p'q'$. They are (a) the inward shift, (b) the outward shift, (c) the shift towards x, and (d) the shift towards y. This distinction overlaps intentionally when one endpoint of the shortcut remains stationary, i.e., when $p = p'$, as in (e), or when $q = q'$, as in (f). For each operation, we provide a sketch on the left, and, on the right, a plot of the positions of the shortcuts along the path from c to a and the path from c to b. In each plot, the region of shortcuts that belong to the same type of movement is shaded.

Proof. Let T be a geometric tree with backbone $\mathcal{B}$, and let $p, q, p', q' \in \mathcal{B}$.

If there exists a diametral pair u, v of $T + pq$ such that $u, v \in T$, and $p'q'$ is indifferent for $\{u, v\}$, i.e., $d_T(u, v) = d_{T+p'q'}(u, v)$, then we have $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$, because

$$\operatorname{diam}(T + pq) = d_{T+pq}(u, v) \leq d_T(u, v) = d_{T+p'q'}(u, v) \leq \operatorname{diam}(T + p'q') \enspace .$$

Therefore, we may assume that $p'q'$ is useful for any diametral pair u, v of $T + pq$ with $u, v \in T$.

If there exists a diametral pair u, v of $T + pq$ with $u, v \in T$ such that $p'q'$ is useful for u, v, the path from u to p' in T contains p, i.e., $d_T(u, p') = d_T(u, p) + d_T(p, p')$, and the path from v to q' in T contains q, i.e., $d_T(q', v) = d_T(q', q) + d_T(q, v)$, then $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$, since

$$\begin{aligned}
\operatorname{diam}(T + pq) &= d_{T+pq}(u, v) && (u, v \text{ is diametral in } T + pq)\\
&\leq d_T(u, p) + |pq| + d_T(q, v)\\
&\leq d_T(u, p) + d_T(p, p') + |p'q'| + d_T(q', q) + d_T(q, v) && (\text{triangle inequality})\\
&= d_T(u, p') + |p'q'| + d_T(q', v) && (p' \text{ lies on the path from } u \text{ to } p)\\
&= d_{T+p'q'}(u, v) && (p'q' \text{ is useful for } u, v)\\
&\leq \operatorname{diam}(T + p'q') \enspace .
\end{aligned}$$

We use this observation to prove that each diametral pair blocks the stated operation.

(1) Suppose $T + pq$ has a diametral pair of type x-y. Let $p'q'$ be a shortcut for T such that the movement from pq to $p'q'$ is an inward shift, as illustrated in Figure 3.17.

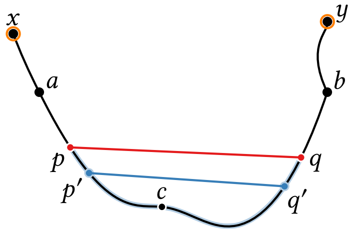


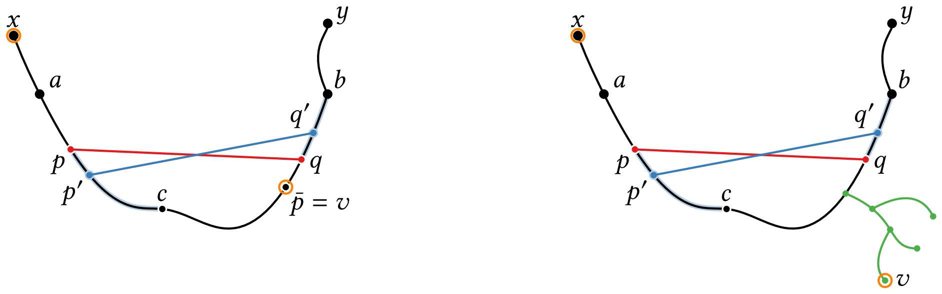
Figure 3.17: An illustration of an inwards shift from pq to $p'q'$.

Then $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$ follows, since x, y is a diametral pair of $T + pq$ with $x, y \in T$ such that p lies on the path from x to p' and q lies on the path from q' to y.

(2) Suppose $T + pq$ has a diametral pair of type x-$\blacksquare$. Let $p'q'$ be a shortcut for T such that the movement from pq to $p'q'$ is a shift towards y, as illustrated in Figure 3.18.

Let x, v be a diametral pair of type x-$\blacksquare$ in $T + pq$. Then $v \in T$, since $|pq| \leq d_T(p, q)$ and, thus, $x, v \in T$. The path from x to p' contains p, since the movement from pq to $p'q'$ is a shift towards y. The path from q' to v in T contains q, since otherwise y would be farther away from x than v in $T + pq$. Therefore, we have $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$.

(3) Suppose $T + pq$ has a diametral pair of type $\blacksquare$-y. Let $p'q'$ be a shortcut for T such that the movement from pq to $p'q'$ is a shift towards x. This case is symmetric to the previous one.

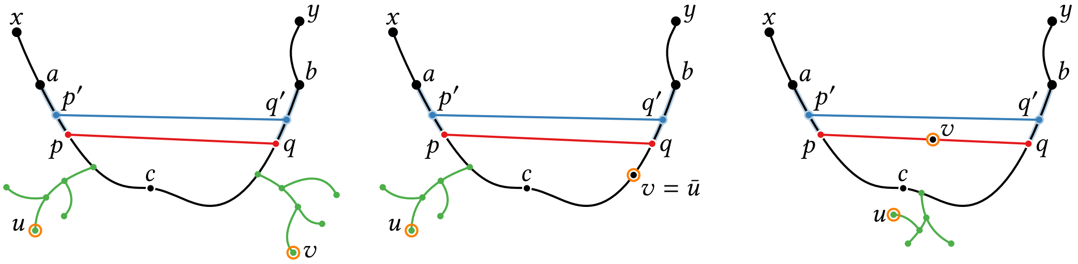


(a) The pair x-■ manifests as x-●.

(b) The pair x-■ manifests as x-▲.

Figure 3.18: An illustration of a y-shift from pq to $p'q'$ that increases the distance between a diametral pair x, v of type x-■ manifesting as (a) subtype x-● and (b) subtype x-▲.

(4) Suppose $T + pq$ has a diametral pair of type ■-■ or ■-○. Let $p'q'$ be a shortcut for T such that the movement from pq to $p'q'$ is an outward shift, as illustrated in Figure 3.19.



(a) A pair of subtype ▲-▲.

(b) A pair of subtype ▲-●.

(c) A pair of subtype ▲-○.

Figure 3.19: An illustration of an outward shift from pq to $p'q'$ with a diametral pair u, v in $T + pq$ manifesting as (a) subtype ▲-▲, (b) subtype ▲-●, and (c) subtype ▲-○.

Every diametral pair of type ■-■ or ■-○ manifests as subtype ▲-▲, ▲-● or ▲-○.

a) Suppose there is some diametral pair u, v of $T + pq$ of subtype ▲-▲.

Then u and v are leaves of two secondary $\mathcal{B}$-sub-trees that are attached to $\mathcal{B}$ along the path from p to q in T. Otherwise, x or y would be strictly farther from v than u.

Then we have $\operatorname{diam}(T + pq) \leq \operatorname{diam}(T + p'q')$, since u, v is a diametral pair of $T + pq$ with $u, v \in T$ where p lies on the path from u to p' and q lies on the path from q' to v.

b) Suppose there is some diametral pair u, v in $T + pq$ of subtype ▲-● or ▲-○.

Then one of u or v is a leaf of a secondary $\mathcal{B}$-sub-tree and the other is the farthest point on $C(p, q)$ from said leaf. Without loss of generality, suppose u lies in a secondary $\mathcal{B}$-sub-tree S. The root r of S must lie along the path from p to q in T. Otherwise, x or y would be strictly farther from u than v. This means we have

$$d_{T+pq}(u, v) = d_T(u, r) + d_{T+pq}(r, v) = d_T(u, r) + \frac{|pq| + d_T(p, q)}{2} \quad .$$

Let v' be the farthest point from u on the simple cycle $C(p'q')$ in $T + p'q'$. The vertex r lies on the path from p' to q' in T, since the movement from pq to $p'q'$ is an outward shift, and since r lies on the path from p to q in T. This means we have

$$d_{T+p'q'}(u, v') = d_T(u, r) + d_{T+p'q'}(r, v') = d_T(u, r) + \frac{|p'q'| + d_T(p', q')}{2} \quad .$$

The simple cycle in the augmented tree is growing as pq shifts outwards to $p'q'$, i.e., $|pq| + d_T(p, q) \leq |p'q'| + d_T(p'q')$. This implies $d_{T+pq}(u, v) \leq d_{T+p'q'}(u, v')$ and, therefore, $\operatorname{diam}(T + pq) = d_{T+pq}(u, v) \leq d_{T+p'q'}(u, v') \leq \operatorname{diam}(T + p'q')$.

We conclude that each type of diametral pair blocks one type of movement as claimed. □

3.3.4 Sudden Optimality

As an immediate consequence of Lemma 3.10, a shortcut pq is optimal for a geometric tree T when each of the four types of movements is blocked by some diametral pair in $T + pq$.

Corollary 3.11. *If an augmented tree $T + pq$ has diametral pairs of the types x-y, x-■, and ■-y, as well as a diametral pair of type ■-■ or ■-○, then pq is an optimal shortcut for T.*

The following result allows us to ignore diametral pairs of type ■-■ when a diametral pair of x-y is present, provided that we keep track of diametral pairs of type ■-y and x-■. This helps us to reach the desired running time, since there may be $O(n^2)$ candidates for pairs of type ■-■.

Theorem 3.12. *If an augmented tree $T + pq$ has diametral pairs of types x-y and ■-■ then $T + pq$ also has diametral pairs of type x-■ and ■-y and, thus, the shortcut pq is optimal for T.*

Proof. Suppose an augmented tree $T + pq$ has diametral pairs of types x-y and ■-■.

Then pq is useful for x, y, since $T + pq$ has diametral pairs besides those of type x-y. Therefore, the diametral pairs of type x-y in $T + pq$ are connected by diametral paths of type x-pq-y.

We distinguish two cases depending on whether the diametral pairs of type ■-■ in $T + pq$ are connected by diametral paths of type ■-pq-■ or ■-T-■. In each case, we argue that there are diametral pairs of types x-■ and of type ■-y, which implies that pq is optimal for T.

1. Suppose $T + pq$ has diametral paths of type x-pq-y and ■-pq-■, as illustrated in Figure 3.20.

   Then $T + pq$ has a diametral pair $u, v \in T \setminus (X \cup Y)$ such that a shortest path from u to v contains pq. We show that the pairs u, y and x, v are diametral in $T + pq$.

   Since pq is useful for u, v, the shortcut pq is also useful for u, q and, thus, for u, y. By comparing the paths x-pq-y and u-pq-y, we obtain $d_T(u, p) \leq d_T(x, p)$, since

$$\begin{aligned} d_T(u, p) + |pq| + d_T(q, y) &= d_{T+pq}(u, y) && (pq \text{ is useful for } u, y) \\ &\leq d_{T+pq}(x, y) && (x, y \text{ is diametral in } T + pq) \\ &= d_T(x, p) + |pq| + d_T(q, y) \quad . && (pq \text{ is useful for } x, y) \end{aligned}$$

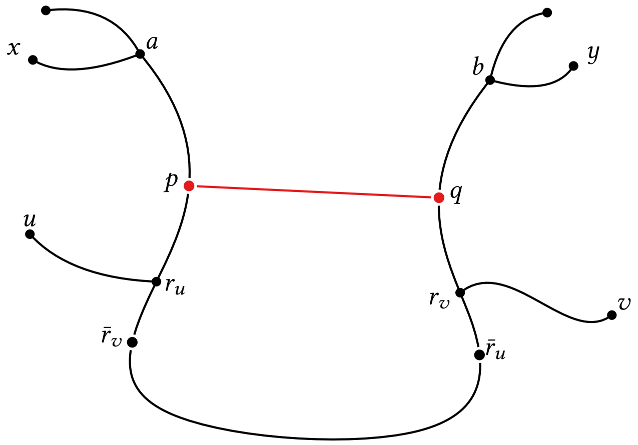


Figure 3.20: An illustration of an augmented tree $T + pq$ with diametral paths of type x-pq-y and ■-pq-■. The diametral path of type ■-pq-■ connects u with v via the shortcut. The points $\bar{r}_u$ and $\bar{r}_v$ mark the farthest points from r_u and r_v, respectively along the simple cycle in $T + pq$. Their position indicates that pq is useful for u, v.

Likewise, we obtain $d_T(q, v) \le d_T(q, y)$ by comparing the paths x-pq-y and x-pq-v.

Comparing the diametral paths x-pq-y and u-pq-v yields $d_T(x, p) + d_T(q, y) = d_T(u, p) + d_T(q, v)$. This equation cannot be satisfied when $d_T(u, p) < d_T(x, p)$ or $d_T(q, v) < d_T(q, y)$, since $d_T(u, p) \le d_T(x, p)$ and $d_T(q, v) \le d_T(q, y)$. Therefore, we have $d_T(u, p) = d_T(x, p)$ and $d_T(q, v) = d_T(q, y)$. This implies that u, y and x, v are diametral pairs in $T + pq$, since

$$d_{T+pq}(u, y) = d_T(u, p) + |pq| + d_T(q, y) = d_T(x, p) + |pq| + d_T(q, y) = \text{diam}(T + pq) \quad ,$$
$$\text{and} \quad d_{T+pq}(x, v) = d_T(x, p) + |pq| + d_T(q, v) = d_T(x, p) + |pq| + d_T(q, y) = \text{diam}(T + pq) \quad .$$

Hence, $T + pq$ is in the pair state $\{x\text{-}y, \blacksquare\text{-}\blacksquare, x\text{-}\blacksquare, \blacksquare\text{-}y\}$ and, thus, pq is optimal for T.

2. Suppose $T + pq$ has diametral paths of type x-pq-y and ■-T-■, as illustrated in Figure 3.21.

   Then there exists a diametral pair u, v of $T + pq$ with $u, v \in T$ such that there is a shortest path from u to v in $T + pq$ that does not contain pq. If u lies in a secondary $\mathcal{B}$-sub-tree S_u, then let r_u be the root of S_u. Otherwise, let $r_u = u$. Likewise, let r_v be the root of the $\mathcal{B}$-sub-tree containing v or let $r_v = v$ when $v \in \mathcal{B}$. We assume, without loss of generality, that r_u lies on the path in T from a to r_v. Otherwise, we swap u and v.

   We show that u, y and x, v are diametral in $T + pq$. Since u-T-v is diametral, pq cannot be useful for u, v and, thus, pq cannot be useful for r_u, r_v, i.e., $d_T(r_u, r_v) \le d_T(r_u, p) + |pq| + d_T(q, r_v)$. On the other hand, pq must be useful for x, v, i.e., $d_{T+pq}(x, v) < d_T(x, v)$, since

$$\begin{aligned} d_{T+pq}(x, v) &\le \text{diam}(T + pq) \\ &= d_{T+pq}(u, v) && (u, v \text{ is diametral in } T + pq) \\ &= d_T(u, v) && (pq \text{ is not useful for } u, v) \end{aligned}$$

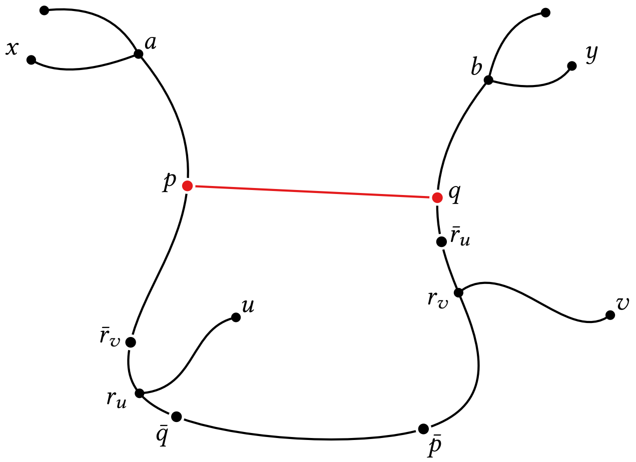


Figure 3.21: An illustration of an augmented tree $T + pq$ with diametral paths of type x-pq-y and $\blacksquare$-T-$\blacksquare$. The diametral path of type $\blacksquare$-T-$\blacksquare$ connects u with v via the tree only. The points $\bar{r}_u$ and $\bar{r}_v$ mark the farthest points from r_u and r_v, respectively along the simple cycle in $T + pq$. Their position indicates that pq is useless for u, v.

$$
\begin{aligned}
&= d_T(u, r_u) + d_T(r_u, v) && (r_u \text{ lies on the path from } u \text{ to } v)\\
&< d_T(x, r_u) + d_T(r_u, v) && (u \in T \setminus X)\\
&= d_T(x, v) \quad . && (r_u \text{ lies on the path from } x \text{ to } v)
\end{aligned}
$$

Likewise, pq must be useful for u, y. We have $d_T(q, v) \leq d_T(q, y)$, as

$$
\begin{aligned}
d_T(x, p) + |pq| + d_T(q, v) &= d_{T+pq}(x, v) && (pq \text{ is useful for } x, v)\\
&\leq d_{T+pq}(x, y) && (x, y \text{ is diametral in } T + pq)\\
&= d_T(x, p) + |pq| + d_T(q, y) && (pq \text{ is useful for } x, y)
\end{aligned}
$$

The pair u, y is diametral in $T + pq$, i.e., $d_{T+pq}(u, y) = \operatorname{diam}(T + pq)$, since

$$
\begin{aligned}
\operatorname{diam}(T + pq) &= d_{T+pq}(u, v) && (u, v \text{ is diametral in } T + pq)\\
&= d_T(u, v) && (pq \text{ is not useful for } u, v)\\
&= d_T(u, r_u) + d_T(r_u, r_v) + d_T(r_v, v)\\
&\leq d_T(u, r_u) + d_T(r_u, p) + |pq| + d_T(q, r_v) + d_T(r_v, v)\\
& && (pq \text{ not useful for } r_u, r_v)\\
&= d_T(u, p) + |pq| + d_T(q, v)\\
&\leq d_T(u, p) + |pq| + d_T(q, y) && (d(q, v) \leq d(q, y))\\
&= d_{T+pq}(u, y) && (pq \text{ is useful for } u, y)\\
&\leq \operatorname{diam}(T + pq) \quad .
\end{aligned}
$$

Likewise, x, v is diametral in $T + pq$, and thus, $T + pq$ is in pair state $\{x\text{-}y, x\text{-}\blacksquare, \blacksquare\text{-}y, \blacksquare\text{-}\blacksquare\}$.

Therefore, if $T + pq$ has diametral pairs of type x-y and $\blacksquare$-$\blacksquare$, then pq is optimal for T. □

3.4 Continuous Algorithm

Inspired by the plane-sweep paradigm, we—conceptually—move the shortcut continuously while changing its speed and direction at certain events, i.e., when the pair state or path state changes. To implement this approach, we discretize this movement such that the shortcut jumps from one event to the next. We discretize the continuous algorithm in the next section.

3.4.1 The Algorithm from the Pair State Perspective

Figure 3.22 describes the continuous algorithm in terms of the pair states and operations. Initially, we place the shortcut with both endpoints on the absolute center c of the geometric tree T. This ensures that we start in pair state $\{x\text{-}y\}$. The algorithm consists of at most three phases: an outwards shift, possibly followed by a shift towards x or a shift towards y, possibly followed by another outwards shift. Some pair states are marked as final states with a double border. If we reach a final state, we terminate our search and report the best shortcut that we have found. For the other states, we specify the direction in which we move the shortcut.

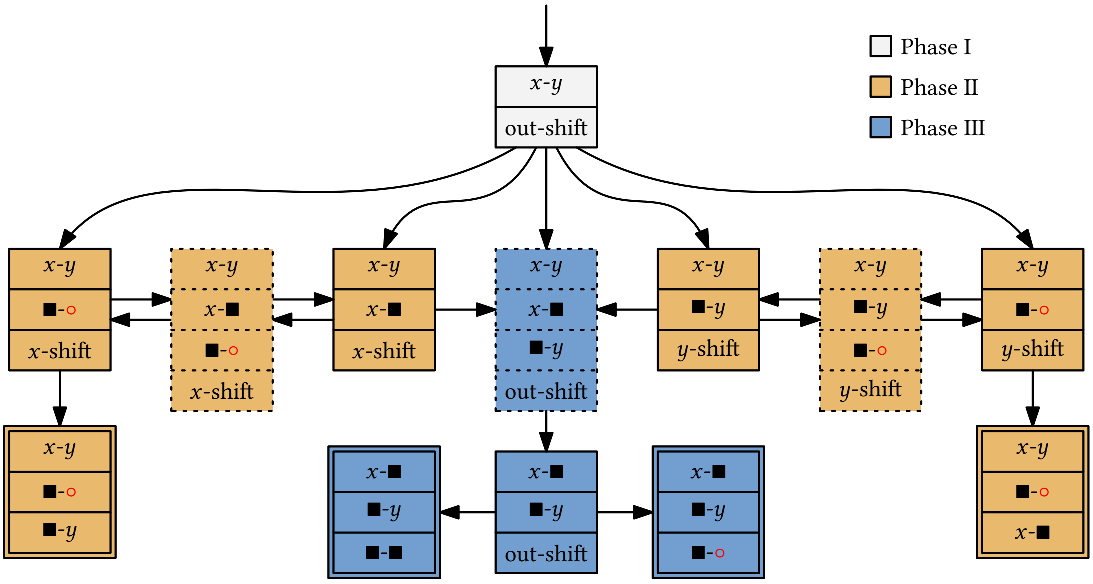


Figure 3.22: The pair states encountered during our search for an optimal shortcut for a tree. There are three types of states: First, regular states (single boundary) indicate the pair state and the operation applied (out-shift, x-shift, or y-shift). Second, transition states (dotted boundary) are visited only momemtarily while transitioning from one regular state to another regular state. Third, final states (double boundary) where we terminate our search and report the best shortcut encountered. Under certain conditions, the search may also terminate early in non-final states. We always start in state $\{x\text{-}y\}$ with an outward shift. When we reach the pair state $\{x\text{-}y$, ■-○$\}$ then we perform both a shift towards x *and* separately a shift towards y.

For the sake of simplicity, we omit some pair states and transitions from Figure 3.22. First, we omit all pair states containing $x\text{-}y$ and $\blacksquare\text{-}\blacksquare$, due to Theorem 3.12. Second, we omit transitions that are implied by transitivity, e.g., we model a transition from $\{x\text{-}y\}$ to $\{x\text{-}y, x\text{-}\blacksquare, \blacksquare\text{-}y\}$ by transitioning from $\{x\text{-}y\}$ to $\{x\text{-}y, x\text{-}\blacksquare\}$ and then from $\{x\text{-}y, x\text{-}\blacksquare\}$ to $\{x\text{-}y, x\text{-}\blacksquare, \blacksquare\text{-}y\}$. Third, we omit pair states that are supersets of any final states in Figure 3.22.

Phase I: Shifting Outwards In Phase I, we shorten all diametral paths of type $x\text{-}y$ with an outwards shift, i.e., we move p from c towards a and we move q from c towards b. If p reaches a before q reaches b, then p remains at a and q continues to move towards b. Likewise, p continues to move towards a if q reaches b. While we are in Phase I, the current shortcut is the best shortcut encountered so far. Phase I ends when the shortcut reaches the end of the backbone, i.e., $pq = ab$, or when a second type of diametral pair appears and Phase II begins.

Phase II: Shifting Sideways The second phase begins when we transition from pair state $\{x\text{-}y\}$ to a pair state containing $x\text{-}y$. If we transition from $\{x\text{-}y\}$ to $\{x\text{-}y, x\text{-}\blacksquare\}$, then we shift towards x. If we transition from $\{x\text{-}y\}$ to $\{x\text{-}y, \blacksquare\text{-}y\}$, then we shift towards y. If we transition from $\{x\text{-}y\}$ to $\{x\text{-}y, \blacksquare\text{-}\circ\}$, then we branch the search into a shift towards x and a shift towards y. In the following, we discuss the second phase for the shift towards x, i.e., for the states $\{x\text{-}y, x\text{-}\blacksquare\}$, $\{x\text{-}y, x\text{-}\blacksquare, \blacksquare\text{-}\circ\}$, and $\{x\text{-}y, \blacksquare\text{-}\circ\}$. The shift towards y for the pair states $\{x\text{-}y, \blacksquare\text{-}y\}$, $\{x\text{-}y, \blacksquare\text{-}y, \blacksquare\text{-}\circ\}$, and $\{x\text{-}y, \blacksquare\text{-}\circ\}$ is symmetric.

Suppose we reach the pair state $\{x\text{-}y, x\text{-}\blacksquare\}$ from $\{x\text{-}y\}$. All diametral paths in $T + pq$ contain the path from a to p. We move p closer to a, thereby shrinking the diameter. At the same time, we move q with a speed towards a that keeps all diametral paths in balance. This ensures that we remain in the current pair state until another diametral pair appears. While we are in this state, the current shortcut is the best shortcut encountered so far.

When we reach the pair state $\{x\text{-}y, \blacksquare\text{-}\circ\}$ then we move p towards a and adjust the position of q to balance the diametral paths of type $x\text{-}pq\text{-}y$ with those of type $\blacksquare\text{-}p\text{-}\circ$ and $\blacksquare\text{-}q\text{-}\circ$. In this state, the diameter shrinks and grows proportional to the length of the shortcut and the best shortcut so far is the shortest shortcut encountered since we entered this state.

Balancing the diametral paths when moving pq ensures that we remain in the current pair state until another diametral pair appears. It also restricts our search considerably: we are essentially conducting a linear search, since the speed of q is determined by the speed of p, the path state, and the change in the length of the shortcut, as we see in Section 3.5.

Phase II ends when p reaches a, when we transition to a pair state containing $x\text{-}y$ and $\blacksquare\text{-}\blacksquare$, when we transition from $\{x\text{-}y, \blacksquare\text{-}\circ\}$ to the final pair state $\{x\text{-}y, \blacksquare\text{-}\circ, \blacksquare\text{-}y\}$, or when we transition from $\{x\text{-}y, x\text{-}\blacksquare\}$ to the pair state $\{x\text{-}y, x\text{-}\blacksquare, \blacksquare\text{-}y\}$ and where we begin Phase III.

Phase III: Shifting Outwards We begin Phase III when we reach the pair state $\{x\text{-}y, x\text{-}\blacksquare, \blacksquare\text{-}y\}$ from $\{x\text{-}y, x\text{-}\blacksquare\}$ or from $\{x\text{-}y, \blacksquare\text{-}y\}$. Since $x\text{-}y$, $x\text{-}\blacksquare$, and $\blacksquare\text{-}y$ block all other movements, we shift outwards balancing $x\text{-}\blacksquare$ and $\blacksquare\text{-}y$. We immediately transition to the pair state $\{x\text{-}\blacksquare, \blacksquare\text{-}y\}$, since the path from x to y via the shortcut shrinks faster than the diametral paths connecting $x\text{-}\blacksquare$ and $\blacksquare\text{-}y$. If we reach Phase III, then the shortest shortcut encountered during Phase III is optimal. Phase III ends when p hits a, when q hits b, or when $\blacksquare\text{-}\blacksquare$ or $\blacksquare\text{-}\circ$ appears.

3.4.2 Optimality

We argue that the shortcut produced by the above algorithm is indeed optimal, using invariants for each pair state that follow from the blocking lemma (Lemma 3.10). Moreover, we show that throughout the course of the algorithm one endpoint of the shortcut remains on the path from a to c while the other endpoint remains on the path from c to b along the backbone.

While the algorithm is in Phase I, there is an optimal shortcut p^*q^* that we reach from the current shortcut pq by shifting outwards, by shifting towards x, by shifting towards y, or by remaining stationary. This invariant holds because the diametral pairs of type x-y block any inward shift, i.e., we have $\operatorname{diam}(T + pq) \le \operatorname{diam}(T + p'q')$ for any shortcut $p'q'$ that we reach with an inward shift from pq, due to Lemma 3.10. Therefore, if Phase I concludes with the shortcut reaching the end of the backbone, i.e., $pq = ab$, then ab is an optimal shortcut for T.

While the algorithm is in pair state $\{x\text{-}y, x\text{-}\blacksquare\}$ of Phase II, there is an optimal shortcut p^*q^* that we reach from the current shortcut pq by shifting towards x, by shifting outwards, or by remaining stationary. This invariant holds because the diametral pairs of type x-y and x-$\blacksquare$ in $T + pq$ block any inward shift and any shift towards y, i.e., we have $\operatorname{diam}(T + pq) \le \operatorname{diam}(T + p'q')$ for any shortcut $p'q'$ that we reach with an inward shift or a shift towards y from pq, due to Lemma 3.10. Therefore, if Phase II concludes with $p = a$ in pair state $\{x\text{-}y, x\text{-}\blacksquare\}$, then the current shortcut is optimal, because when $p = a$, every shift towards x is also an inwards shift and, thus, blocked by x-y, and every outwards shift is also a shift towards y and, thus, blocked by x-$\blacksquare$.

While the algorithm is in pair state $\{x\text{-}y, \blacksquare\text{-}\circ\}$ of Phase II, there is an optimal shortcut p^*q^* that we reach from the current shortcut pq by shifting towards x, by shifting towards y, or by remaining stationary. This invariant holds because the diametral pairs of type x-y and $\blacksquare$-$\circ$ in $T + pq$ block any inward shift and any outward shift, i.e., we have $\operatorname{diam}(T + pq) \le \operatorname{diam}(T + p'q')$ for any shortcut $p'q'$ that we reach with an inward or outward shift from pq, due to Lemma 3.10. The following invariant shows that we cannot miss an optimal shortcut in this state.

Invariant 3.13. *Suppose there is an optimal shortcut p^*q^* for T in the direction of a shift towards x when the algorithm transitions to the pair state $\{x\text{-}y, \blacksquare\text{-}\circ\}$ for the first time. While the algorithm shifts the shortcut towards x in pair state $\{x\text{-}y, \blacksquare\text{-}\circ\}$ of Phase II, we have already encountered an optimal shortcut or the shift from the current shortcut pq to p^*q^* is still a shift towards x.*

Proof. Suppose we shift the shortcut from pq towards x while maintaining the diametral paths x-y and $\blacksquare$-$\circ$ in balance until the pair state changes at some position $p'q'$. If the movement from $p'q'$ to p^*q^* is a shift towards x, then so is the movement from pq to p^*q^*, by transitivity.

Suppose the shift from pq to p^*q^* leads towards x while the shift from $p'q'$ to p^*q^* leads towards y. We argue that we encounter an optimal shortcut while we are shifting from pq to $p'q'$. Let $p''q''$ be the last position during the shift from pq to $p'q'$ where the shift from $p''q''$ to p^*q^* leads towards x. Then $T + p''q''$ is in pair state $\{x\text{-}y, \blacksquare\text{-}\circ\}$ and we have $p'' = p^*$ or $q'' = q^*$. If $p'' = p^*$, then the movement from $p''q''$ to p^*q^* is an inward shift towards x. Since x-y blocks any inward shift, we have $\operatorname{diam}(T + p''q'') \le \operatorname{diam}(T + p^*q^*)$. If $q'' = q^*$, then the movement from $p''q''$ to p^*q^* is an outward shift towards x. Since $\blacksquare$-$\circ$ blocks any outward shift, we have $\operatorname{diam}(T + p''q'') \le \operatorname{diam}(T + p^*q^*)$. In both cases, $p''q''$ is optimal, as p^*q^* is optimal. □

If Phase II ends in state $\{x\text{-}y, \blacksquare\text{-}\circ\}$ with $p = a$ or in the final state $\{x\text{-}y, \blacksquare\text{-}\circ, \blacksquare\text{-}y\}$, then all directions are blocked and, by the invariant, we have encountered an optimal shortcut.

When the algorithm enters Phase III with a transition to the pair state $\{x\text{-}y, x\text{-}\blacksquare, \blacksquare\text{-}y\}$, then there is an optimal shortcut p^*q^* that we reach from the current shortcut pq by shifting outwards or by remaining stationary. This is because all movements, except for the outward shift, are blocked by the diametral pairs of types $x\text{-}y$, $x\text{-}\blacksquare$, and $\blacksquare\text{-}y$. As we shift the shortcut outwards in the pair state $\{x\text{-}\blacksquare, \blacksquare\text{-}y\}$ of Phase III, we have already encountered an optimal shortcut or the optimal shortcut p^*q^* can still be reached with an outward shift from the current shortcut.

Therefore, we have encountered an optimal shortcut when Phase III ends in the final state $\{x\text{-}\blacksquare, \blacksquare\text{-}y, \blacksquare\text{-}\blacksquare\}$, or in $\{x\text{-}\blacksquare, \blacksquare\text{-}y, \blacksquare\text{-}\circ\}$, or when $p = a$ or $q = b$ in the state $\{x\text{-}\blacksquare, \blacksquare\text{-}y\}$.

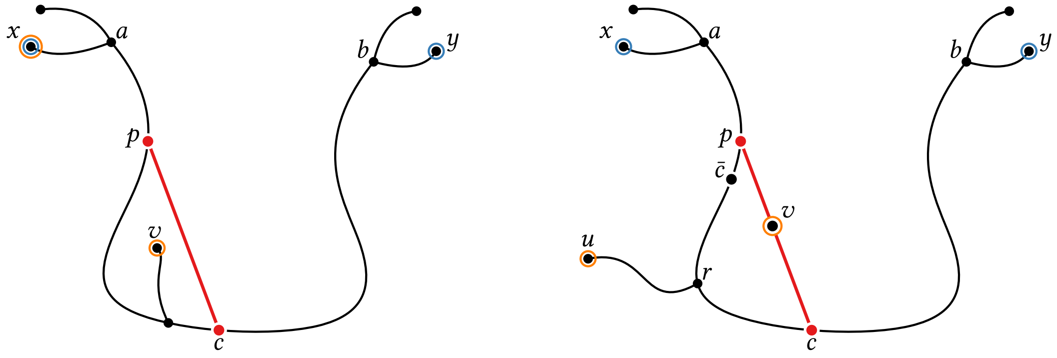
Finally, we argue that the endpoints of the shortcut remain on their respective sub-paths of the backbone, i.e., that the absolute center c always lies on the path from p to q.

Invariant 3.14. *At any moment during the course of the continuous algorithm, the point p lies on the path from a to c in T and the point q lies on the path from c to b in T.*

Proof. The invariant holds when we place both p and q at the absolute center c at the beginning. Throughout Phase I, we perform an outwards shift that continues to uphold the invariant: both p and q move away from c and neither does p move past a nor does q move past b.

For Phase II, we prove that q cannot pass through c during a shift towards x and, symmetrically, that p cannot pass through c during a shift towards y. This means that the invariant holds throughout Phase II, since the algorithm terminates when p reaches a or when q reaches b.

Assume for the sake of a contradiction that q reaches c during a shift towards x in Phase II. Without loss of generality, suppose that this is the first time q reaches c during Phase II. This means the invariant holds until now and, therefore, p lies on the path from a to c. Since we are in Phase II, there are diametral pairs of type $x\text{-}y$ in $T + pq$ as well as diametral pairs of type $x\text{-}\blacksquare$ or $\blacksquare\text{-}\circ$. We distinguish two cases depending on the pair state of the augmented tree $T + pc$. In each case, we derive a contradiction, meaning that q could never have reached c.



(a) Path state $\{x\text{-}y, x\text{-}\blacksquare\}$ for $T + pc$.

(b) Path state $\{x\text{-}y, \blacksquare\text{-}\circ\}$ for $T + pc$.

Figure 3.23: Two impossible constellations where an augmented tree $T + pc$ has (a) diametral pairs of types $x\text{-}y$ and $x\text{-}\blacksquare$ or (b) diametral pairs of type $x\text{-}y$ and $\blacksquare\text{-}\circ$.

1. Suppose $T + pc$ has diametral pairs of type x-y and x-■, as illustrated in Figure 3.23a.

   Let x, v be a diametral pair of type x-■ in $T + pc$. Since v does not lie in a primary $\mathcal{B}$-sub-tree of T, we have $d_T(c, v) < d_T(c, y)$. Since $T + pc$ has other diametral pairs besides those of type x-y, the shortcut pc must be useful for x, y, i.e., $d_T(x, p) + |pc| + d_T(c, y) = d_{T+pc}(x, y)$.

   This leads to the contradiction $\operatorname{diam}(T + pc) < \operatorname{diam}(T + pc)$, since

   $$\begin{aligned}\operatorname{diam}(T + pc) &= d_{T+pc}(x, v) && (x, v \text{ is diametral in } T + pc)\\ &\leq d_T(x, p) + |pc| + d_T(c, v) &&\\ &< d_T(x, p) + |pc| + d_T(c, y) && (d_T(c, v) < d_T(c, y))\\ &= d_{T+pc}(x, y) && (pc \text{ is useful for } x, y)\\ &= \operatorname{diam}(T + pc)\ . && (x, y \text{ is diametral in } T + pc)\end{aligned}$$

2. Suppose $T + pc$ has diametral pairs of type x-y and ■-○, as illustrated in Figure 3.23b.

   Let u, v be a diametral pair of type ■-○ in $T + pc$ with $v \notin T$. Then $v \in pc$ with $p \neq v \neq c$ and u lies in a secondary $\mathcal{B}$-sub-tree with root r attached to the path from p to c along $\mathcal{B}$. Let $\bar{c}$ be the farthest point from c on the simple cycle $C(p, c)$ in $T + pc$.

   This leads to the contradiction $\operatorname{diam}(T + pc) < \operatorname{diam}(T + pc)$, since

   $$\begin{aligned}\operatorname{diam}(T + pc) &= d_{T+pc}(u, v) && (u, v \text{ is diametral in } T + pc)\\ &= d_T(u, r) + d_{T+pc}(r, v) && (u, v \text{ is of type } ■\text{-}○)\\ &\leq d_T(u, c) + d_{T+pc}(r, v) && (r \text{ is on the path from } p \text{ to } c)\\ &< d_T(y, c) + d_{T+pc}(r, v) && (u \text{ is in a secondary } \mathcal{B}\text{-sub-tree})\\ &= d_T(y, c) + (|pc| + d_T(p, c))/2 && (r \text{ and } v \text{ are antipodal along } C(p, c))\\ &= d_T(y, c) + d_{T+pc}(c, \bar{c}) && (c \text{ and } \bar{c} \text{ are antipodal along } C(p, c))\\ &= d_{T+pc}(y, \bar{c}) && (c \text{ is on any path from } y \text{ to } \bar{c} \text{ in } T + pc)\\ &\leq \operatorname{diam}(T + pc)\ . &&\end{aligned}$$

Therefore, during Phase II, q remains on the path from c to b and, likewise, p remains on the path from a to c. Finally, this invariant holds for the entire algorithm, because the outward shift of Phase III cannot move p or q through c and it ends when p reaches a or when q reaches b. ☐

In conclusion, if we could implement and run it, the continuous algorithm would produce an optimal shortcut pq for T where the point p lies along the path from a to c and the point q lies along the path from c to b. Next, we simulate the continuous algorithm with a discretization.

3.5 Discretization

To discretize the continuous algorithm, we subdivide the continuous motion of the shortcut with events such that we can calculate the next event and the change in the continuous diameter of $T + pq$ between subsequent events. We introduce events, for instance, when the shortcut hits a vertex, when the path state changes, and when the shortcut begins to shrink or to grow.

3.5.1 Simulating Phase I

In the continuous algorithm, we move p and q with unit speed towards a and b, respectively. In the discrete algorithm, we process a vertex event whenever p or q would hit a vertex during the continuous movement. We locate the next vertex event by comparing the distance from p and from q to the next vertex along their respective paths towards a and towards b.

Phase I begins in path state $\{x\text{-}pq\text{-}y, x\text{-}T\text{-}y\}$ with $pq = cc$; the diametral paths of type x-T-y disappear when the shortcut becomes useful for T. During Phase I, the continuous diameter decreases or remains constant. Therefore, it is sufficient for the discrete algorithm to determine where Phase I of the continuous algorithm ends. At the end of Phase I, we have either reached the end of the backbone, i.e., $pq = ab$, or a diametral pair of type x-$\blacksquare$, $\blacksquare$-y, or $\blacksquare$-$\circ$ has appeared alongside the diametral pairs of type x-y. We may ignore diametral pairs of type $\blacksquare$-$\blacksquare$ as they appear together with x-$\blacksquare$ and $\blacksquare$-y when x-y is diametral, as shown in Theorem 3.12. We detect changes in the path state by monitoring the candidates for each type of diametral path—except for those connecting diametral pairs of type $\blacksquare$-$\blacksquare$.

x-pq-y The path x-pq-y has length $d_{T+pq}(x, y) = d_T(x, p) + |pq| + d_T(q, y)$. Between two events, p and q remain on their respective containing edge. If the edges of T are line segments, we can express the positions of p and q as an algebraic function of time with constant degree. Therefore, we can also express $d_{T+pq}(x, y)$ as an algebraic function of small degree.

x-pq-$\blacktriangle$, $\blacktriangle$-pq-y If the path x-pq-s_i, for $i = 1, 2, \ldots, k$ becomes a diametral path of type x-pq-$\blacktriangle$, then pq is useful or indifferent for x, s_i, i.e., the leaf s_i belongs to a secondary $\mathcal{B}$-sub-tree S_i that is attached to the path from $\bar{p}$ to b in T. If x-pq-s_i becomes diametral in Phase I or II, then x-pq-y is also diametral and S_i is attached to the path from q to $\bar{p}$, as otherwise y would be farther from x than s_i in $T + pq$. Therefore, for any $i = 1, 2, \ldots, k$, the paths x-pq-y and x-pq-s_i are diametral in $T + pq$ if and only if pq is useful or indifferent for x, s_i and $d_T(q, y) = d_T(q, s_i)$, i.e., q lies midway along the path from s_i to y.

We introduce new events at the points $q_i \in T$ with $d_T(q_i, y) = d_T(q_i, s_i)$, for $i = 1, 2, \ldots, k$, whenever q_i lies on the path from c to b. To locate the points $q_1, q_2, \ldots, q_k$, we first sort the distances $d_T(y, s_1), d_T(y, s_2), \ldots, d_T(y, s_k)$ and then traverse the path from b to c placing q_i at the appropriate distance from y. This takes $O(n + k \log k) = O(n \log n)$ time.

When q reaches q_i, for some $i = 1, 2, \ldots, k$, during Phase I we test whether pq is useful or indifferent for x, s_i. If pq is useful or indifferent for x, s_i, then x-pq-s_i becomes diametral and Phase I ends. Otherwise, pq is useless for x, s_i and x-pq-s_i cannot become diametral during Phase I. Processing this event takes constant time, because the shortcut pq is useful or indifferent for x, s_i if and only if the root r_i of S_i lies on the path from $\bar{p}$ to b in $T + pq$, which we can check in constant time by comparing $d_T(a, r_i)$ with $d_T(a, \bar{p})$, since

$$d_T(a, \bar{p}) = d_T(a, p) + d_T(p, \bar{p}) = d_T(a, p) + \frac{|pq| + d_T(p, q)}{2} \quad .$$

x-T-$\blacktriangle$, $\blacktriangle$-T-y The length of the path x-T-s_i for some $i = 1, 2, \ldots k$ does not change with the position of pq; what does change, however, is whether x-T-s_i is a *shortest* path in $T + pq$ or not: the shortcut pq is useful for x, s_i if and only if $\bar{p}$ lies in the interior of the path from a to r_i in T. Otherwise, pq is indifferent for $\{x, s_i\}$, i.e., $d_{T+pq}(x, s_i) = d_T(x, s_i)$.

The path x-T-s_i becomes a diametral path of type x-T-▲ when the continuous diameter of $T + pq$ decreases to $d_T(x, s_i)$ while the shortcut pq is indifferent for $\{x, s_i\}$. To detect this, we sort the values $d_T(x, s_1), d_T(x, s_2), \ldots, d_T(x, s_k)$ and we introduce events when the current continuous diameter reaches any of these values. This leads to $O(n \log n)$ additional preprocessing time and $O(k) = O(n)$ additional events, since the continuous diameter is decreasing during Phase I. For each additional event, we compare $d_T(a, r_i)$ with $d_T(a, \bar{p})$ to decide, in constant time, whether pq is indifferent for $\{x, s_i\}$.

x-pq-●, ●-pq-y, x-T-●, ●-T-y, ▲-p-○, ▲-q-○, ▲-pq-●, ▲-T-● We need to detect diametral paths in $T + pq$ that connect some leaf l of T with the farthest point from l along the cycle $C(p, q)$. Any candidate for a diametral path of this kind has length $h + \frac{1}{2}(d_T(p, q) + |pq|)$, where h is the height of a tallest sub-tree attached to the cycle $C(p, q)$ in $T + pq$.

At the beginning of Phase I, the sub-trees containing x and y are the tallest sub-trees attached to $C(p, q)$, i.e., $h = d_T(x, p) = d_T(q, y)$. As Phase I progresses, these sub-trees shrink as p moves to a and q moves to b. For any secondary $\mathcal{B}$-sub-tree S_i attached to the path from a to p, we have $d_T(r_i, s_i) \le d_T(p, s_i) < d_T(p, x)$, and for every secondary $\mathcal{B}$-sub-tree S_j attached to the path from q to b, we have $d_T(r_j, s_j) \le d_T(q, s_j) < d_T(q, y)$. Thus, a secondary $\mathcal{B}$-sub-tree S_i may only become a tallest sub-tree if its root r_i lies on the path from p to q and $h = \max\{d_T(x, p), d_T(r_1, s_1), d_T(r_2, s_2), \ldots, d_T(r_k, s_k), d_T(q, y)\}$.

We compute $\hat{h} = \max\{d_T(r_1, s_1), d_T(r_2, s_2), \ldots, d_T(r_k, s_k)\}$ as part of our preprocessing. This allows us to detect diametral paths of type x-pq-●, ●-pq-y, x-T-●, ●-T-y, ▲-p-○, ▲-q-○, ▲-pq-●, and ▲-T-● by comparing $\max\{d_T(x, p), \hat{h}, d_T(q, y)\} + \frac{1}{2}(d_T(p, q) + |pq|)$ with the current continuous diameter. This takes constant additional time per event.

In conclusion, we can simulate Phase I of the continuous algorithm in $O(n \log n)$ time.

3.5.2 Simulating Phase II

We describe the discretization of Phase II for a shift towards x, where the continuous algorithm balances the diametral path x-pq-y and a diametral path with endpoints of type x-■ or ■-○. Suppose p moves with unit speed towards a. The following lemma specifies the speed at which q should move towards c to balance the diametral paths and how this impacts the diameter.

Lemma 3.15. *Let pq and $p'q'$ be two shortcuts for a geometric tree T such that the movement from pq to $p'q'$ is a shift towards x. If $T + pq$ and $T + p'q'$ are in the same path state, then we can express the distance of q and q' and the change in diameter as stated in Table 3.1.*

Proof. We show the result for the path state $\{x$-pq-y, x-pq-●, x-T-●$\}$. Suppose the paths x-pq-y, x-pq-●, and x-T-● remain diametral as the shortcut moves from pq to $p'q'$. Then

$$d_{T+pq}(x, y) - d_{T+p'q'}(x, y) = \operatorname{diam}(T + pq) - \operatorname{diam}(T + p'q') = d_{T+pq}(x, \bar{x}) - d_{T+p'q'}(x, \bar{x}')$$

Path State	$\boldsymbol{d_T(q,q')}$	$\mathbf{diam}\boldsymbol{(T+pq)} - \mathbf{diam}\boldsymbol{(T+p'q')}$
$\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}\blacktriangle\}$	0	$d_T(p',p) + \lvert pq\rvert - \lvert p'q'\rvert$
$\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}\bullet, x\text{-}T\text{-}\bullet\}$	$\frac{1}{3}\left(d_T(p,p') + \lvert pq\rvert - \lvert p'q'\rvert\right)$	$\frac{2}{3}\left(d_T(p,p') + \lvert pq\rvert - \lvert p'q'\rvert\right)$
$\{x\text{-}pq\text{-}y, x\text{-}T\text{-}\blacktriangle\}$	$d_T(p,p') + \lvert pq\rvert - \lvert p'q'\rvert$	0
$\{x\text{-}pq\text{-}y, \blacktriangle\text{-}p\text{-}\circ, \blacktriangle\text{-}q\text{-}\circ\}$	$d_T(p,p') + \frac{1}{3}\left(\lvert pq\rvert - \lvert p'q'\rvert\right)$	$\frac{2}{3}\left(\lvert pq\rvert - \lvert p'q'\rvert\right)$

Table 3.1: The distance between q and q' and the change in diameter when shifting the shortcut towards x from pq to $p'q'$ while maintaining the diametral paths in balance.

and, thus, $d_T(q,q') = \frac{1}{3}\left(d_T(p,p') + |pq| - |p'q'|\right)$, since

$$\begin{aligned} d_{T+pq}(x,y) - d_{T+p'q'}(x,y) &= d_{T+pq}(x,\bar{x}) - d_{T+p'q'}(x,\bar{x}') \\ \Rightarrow \left(d_T(p',p) + |pq| - |p'q'| + d_T(q,q')\right)/2 &= d_T(p',p) + |pq| - |p'q'| - d_T(q',q) \\ \Rightarrow 3d_T(q,q') &= d_T(p,p') + |pq| - |p'q'| \ . \end{aligned}$$

In this case, the diameter changes by $\frac{2}{3}\left(d_T(p,p') + |pq| - |p'q'|\right)$, since

$$\begin{aligned} \operatorname{diam}(T+pq) - \operatorname{diam}(T+p'q') &= d_{T+pq}(x,y) - d_{T+p'q'}(x,y) \\ &= d_T(p',p) + |pq| - |p'q'| - d_T(q',q) \\ &= \frac{2}{3}\left(d_T(p,p') + |pq| - |p'q'|\right) \ . \end{aligned}$$

Analogously, we establish the results for the other path states listed in Table 3.1. □

When p and q traverse a fixed pair of edges while shifting towards x, Lemma 3.15 allows us to compute the next vertex event and how the diameter changes between subsequent events. While simulating Phase II, we encounter $O(n)$ events where the shortcut hits a vertex or where the shortcut begins to shrink or grow, due to the following. The shortcut enters each edge at most once, since p and q never change direction (we have $d_T(p,p') > 0$ and $d_T(q,q') \geq 0$ from Lemma 3.15). Thus, we encounter $O(n)$ pairs of edges. Moreover, the shortcut changes at most once between growing and shrinking when both endpoints move along a fixed pair of edges.

Lemma 3.16. *For a geometric tree with n vertices, the path state changes $O(n)$ times in Phase II.*

Proof. In Phase II, the speed of p and q is determined by two or three different types of diametral paths, i.e., by x-pq-y and x-pq-$\blacktriangle$, or by x-pq-y and x-pq-$\bullet$ and x-T-$\bullet$, or x-pq-y and x-T-$\blacktriangle$, or by x-pq-y and $\blacktriangle$-p-$\circ$ and $\blacktriangle$-q-$\circ$. For any path state $\mathfrak{S}$ during Phase II, only the subset $\mathfrak{S}'$ of $\mathfrak{S}$ that leads to the least increase in the continuous diameter determines the speed of p and q. The other types of paths cease to be diametral instantaneously, i.e., we transition from $\mathfrak{S}$ to $\mathfrak{S}'$.

Some transitions between the path states in Phase II are impossible. For instance, suppose we shift pq towards x to $p'q'$ such that the path state remains $\{x\text{-}pq\text{-}y, \blacktriangle\text{-}p\text{-}\circ, \blacktriangle\text{-}q\text{-}\circ\}$ until excluding $p'q'$. Then $T + p'q'$ cannot be in path state $\{x\text{-}pq\text{-}y, \blacktriangle\text{-}p\text{-}\circ, \blacktriangle\text{-}q\text{-}\circ, x\text{-}pq\text{-}\bullet, x\text{-}T\text{-}\bullet\}$, due to the following. From Table 3.1 we know $d_T(q,q') = d_T(p,p') + \frac{1}{3}\left(|pq| - |p'q'|\right)$. As pq moves to $p'q'$, the length of the paths x-pq-$\bullet$ and x-T-$\bullet$ changes by $d_T(p',p) + \frac{2}{3}\left(|pq| - |p'q'|\right)$, since

$$d_{T+pq}(x,\bar{x}) - d_{T+p'q'}(x,\bar{x}') = \left(d_T(p',p) + |pq| - |p'q'| + d_T(q,q')\right)/2$$

$$
\begin{aligned}
&= \frac{1}{2}\left(d_T(p',p) + |pq| - |p'q'| + d_T(p,p') + \frac{1}{3}\left(|pq| - |p'q'|\right)\right) \\
&= d_T(p',p) + \frac{2}{3}\left(|pq| - |p'q'|\right) \quad .
\end{aligned}
$$

Since $d_T(p,p') > 0$, the paths of type $x\text{-}pq\text{-}\bullet$ and $x\text{-}T\text{-}\bullet$ shrink at a faster rate than the diametral paths of type $x\text{-}pq\text{-}y$, $\blacktriangle\text{-}p\text{-}\circ$, and $\blacktriangle\text{-}q\text{-}\circ$. Therefore, $x\text{-}pq\text{-}\bullet$ and $x\text{-}T\text{-}\bullet$ cannot become diametral when the shortcut reaches $p'q'$. Figure 3.24 illustrates the transitions between the path states that may occur during Phase II. We discuss some restrictions on these transitions.

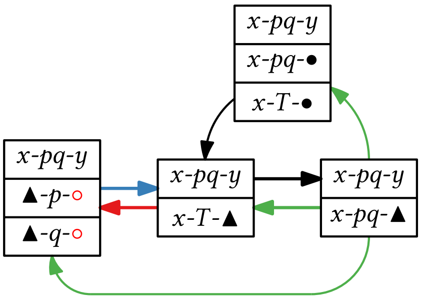


Figure 3.24: The transitions between the path states during Phase II. Transitory states, such as $\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}\blacktriangle, x\text{-}pq\text{-}\bullet, x\text{-}T\text{-}\bullet\}$ have been omitted. The red transition is possible when the shortcut grows and the blue transition is possible when the shortcut shrinks. When we take any green transition from $\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}s_j\}$ for $j = 1, 2, \ldots, k$ then x, s_j ceases to be diametral and cannot become diametral again.

We bound the number of visits to the path state $\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}\blacktriangle\}$ by k, where $k = O(n)$ is the number of secondary $\mathcal{B}$-sub-trees of T. When we are in path state $\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}s_j\}$ for some $j = 1, 2, \ldots, k$, then q lies midway along the path from y to s_j, i.e., $d_T(q,y) = d_T(q,s_j)$. When we transition from $\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}s_j\}$ to any other path state, q will begin to move with non-zero speed towards a. Thus, the pair x, s_j ceases to be diametral and cannot become diametral again during Phase II. Therefore, we may take at most k green transitions in Figure 3.24.

We bound the number of visits to the pair state $\{x\text{-}pq\text{-}y, \blacktriangle\text{-}p\text{-}\circ, \blacktriangle\text{-}q\text{-}\circ\}$ by n. Once we have exhausted the green transitions, we can only enter pair state $\{x\text{-}pq\text{-}y, \blacktriangle\text{-}p\text{-}\circ, \blacktriangle\text{-}q\text{-}\circ\}$ with the red transition from $\{x\text{-}pq\text{-}y, x\text{-}T\text{-}\blacktriangle\}$. This transition is only possible when the shortcut is growing. Therefore, we can visit $\{x\text{-}pq\text{-}y, \blacktriangle\text{-}p\text{-}\circ, \blacktriangle\text{-}q\text{-}\circ\}$ at most $O(n)$ times, since the shortcut switches at most $2n$ times between shrinking and growing during a shift towards x.

Thus, after $O(n)$ visits to each of the path states $\{x\text{-}pq\text{-}y, x\text{-}pq\text{-}\blacktriangle\}$ and $\{x\text{-}pq\text{-}y, \blacktriangle\text{-}p\text{-}\circ, \blacktriangle\text{-}q\text{-}\circ\}$, we can no longer take any of the red or green transitions in Figure 3.24. We encounter $O(n)$ path state changes during Phase II, since the remaining transitions form an acyclic digraph. ☐

In conclusion, we can simulate Phase II in $O(n \log n)$ time, because Phase II consists of $O(n)$ events that we can process in $O(\log n)$ time after $O(n \log n)$ preprocessing time by monitoring the diametral paths of the augmented tree $T + pq$ in the same fashion as in Phase I.

3.5.3 Simulating Phase III

In Phase III, the continuous algorithm balances a diametral path with endpoints of type x-■ and a diametral path with endpoints of type ■-y. This leads to the following speeds.

Lemma 3.17. *Let pq and $p'q'$ be two shortcuts for a geometric tree T such that the movement from pq to $p'q'$ is an outwards shift. If $T+pq$ and $T+p'q'$ are in the same path state, then we can express the distance of q and q' and the change in diameter as stated in Table 3.2.*

x-■	**■-y**	$d_T(q,q')$	$\operatorname{diam}(T+pq)-\operatorname{diam}(T+p'q')$
x-pq-▲	▲-pq-y	$d_T(p,p')$	$\lvert pq\rvert-\lvert p'q'\rvert$
x-pq-▲	●-y	$d_T(p,p')+\frac{1}{3}(\lvert pq\rvert-\lvert p'q'\rvert)$	$\frac{2}{3}(\lvert pq\rvert-\lvert p'q'\rvert)$
x-pq-▲	▲-T-y	$d_T(p,p')+\lvert pq\rvert-\lvert p'q'\rvert$	0
x-●	▲-pq-y	$d_T(p,p')-\frac{1}{3}(\lvert pq\rvert-\lvert p'q'\rvert)$	$\frac{2}{3}(\lvert pq\rvert-\lvert p'q'\rvert)$
x-●	●-y	$d_T(p,p')$	$\frac{1}{2}(\lvert pq\rvert-\lvert p'q'\rvert)$
x-●	▲-T-y	$d_T(p,p')+\lvert pq\rvert-\lvert p'q'\rvert$	0
x-T-▲	▲-pq-y	$d_T(p,p')-(\lvert pq\rvert-\lvert p'q'\rvert)$	0
x-T-▲	●-y	$d_T(p,p')-(\lvert pq\rvert-\lvert p'q'\rvert)$	0
x-T-▲	▲-T-y	$d_T(p,p')$	0

Table 3.2: The distance between q and q' with the change in diameter when shifting a shortcut pq outwards to a new position $p'q'$ while maintaining the diametral paths in balance. Here, x-● stands for x-pq-● and x-T-●. Likewise, ●-y stands for ●-pq-y and ●-T-y.

Proof. We establish these results using the same approach as in Lemma 3.15.

As pq moves to $p'q'$, the potential diametral paths change as follows. As usual, we denote the farthest point from $z\in\{x,y\}$ along $C(p,q)$ and along $C(p',q')$ by $\bar{z}$ and $\bar{z}'$, respectively.

$$
\begin{array}{rl}
x\text{-}pq\text{-}\blacktriangle: & d_{T+pq}(x,s_i)-d_{T+p'q'}(x,s_i)=d_T(p',p)+|pq|-|p'q'|-d_T(q',q)\\
x\text{-}pq\text{-}\bullet,\, x\text{-}T\text{-}\bullet: & d_{T+pq}(x,\bar{x})-d_{T+p'q'}(x,\bar{x}')=\left(d_T(p',p)+|pq|-|p'q'|-d_T(q',q)\right)/2\\
x\text{-}T\text{-}\blacktriangle: & d_{T+pq}(x,s_j)-d_{T+p'q'}(x,s_j)=0\\
\blacktriangle\text{-}pq\text{-}y: & d_{T+pq}(s_l,y)-d_{T+p'q'}(s_l,y)=d_T(q',q)+|pq|-|p'q'|-d_T(p',p)\\
\bullet\text{-}pq\text{-}y,\, \bullet\text{-}T\text{-}y: & d_{T+pq}(y,\bar{y})-d_{T+p'q'}(y,\bar{y}')=\left(d_T(q',q)+|pq|-|p'q'|-d_T(p',p)\right)/2\\
\blacktriangle\text{-}T\text{-}y: & d_{T+pq}(s_o,y)-d_{T+p'q'}(s_o,y)=0
\end{array}
$$

These changes equate for those paths that remain diametral during the shift towards x. For instance, if the paths x-pq-●, x-T-●, and ▲-pq-y remain diametral as pq moves to $p'q'$ then

$$d_{T+pq}(x,\bar{x})-d_{T+p'q'}(x,\bar{x}')=\operatorname{diam}(T+pq)-\operatorname{diam}(T+p'q')=d_{T+pq}(s_l,y)-d_{T+p'q'}(s_l,y)$$

and, thus, $d_T(p,p')=d_T(q,q')+\frac{1}{3}(|pq|-|p'q'|)$, since

$$
\begin{aligned}
&d_{T+pq}(x,\bar{x})-d_{T+p'q'}(x,\bar{x}')=d_{T+pq}(s_l,y)-d_{T+p'q'}(s_l,y)\\
\Rightarrow\ &\left(d_T(p',p)+|pq|-|p'q'|-d_T(q',q)\right)/2=d_T(q',q)+|pq|-|p'q'|-d_T(p',p)
\end{aligned}
$$

$$\Rightarrow 3d_T(p',p) = 3d_T(q,q') + |pq| - |p'q'| \ .$$

In this case, the diameter changes by $\frac{2}{3}(|pq| - |p'q'|)$, since

$$\begin{aligned} \operatorname{diam}(T+pq) - \operatorname{diam}(T+p'q') &= d_{T+pq}(s_l,y) - d_{T+p'q'}(s_l,y) \\ &= d_T(q',q) + |pq| - |p'q'| - d_T(p',p) \\ &= \frac{2}{3}(|pq| - |p'q'|) \ . \end{aligned}$$

In the same manner, we express $d_T(q,q')$ and the change in diameter in terms of $d_T(p,p')$ and $|pq| - |p'q'|$ for the remaining path states listed in Table 3.2. □

Similar to Phase I and II, there are $O(n)$ events where the shortcut hits a vertex or starts to shrink or grow. Even though we can rule out certain transitions between path states, the path state might change $\Omega(n^2)$ times. This occurs, for instance, when the shortcut alternates $\Omega(n)$ times between growing and shrinking such that there are $\Omega(n)$ candidates for diametral pairs, for each of which the shortcut becomes useful whenever it shrinks and useless whenever it grows. We circumvent this issue by ignoring certain superfluous path state events.

Suppose that the shortcut is growing while the path x-T-s_i, for some $i = 1, 2, \ldots, k$, is a diametral path of type x-T-▲. In this situation a path state change may occur where the path x-pq-$\bar{x}$ or a path x-pq-s_j for some $j > i$ might become a diametral path of type x-pq-● and x-pq-▲, respectively. There is no need to recognize this path state change, since the diameter cannot decrease before the shortcut becomes useful for x, s_i again. Until then, we do not keep track of any changes in the diametral path for pairs of type x-■. With this modification, the shortcut might leave the trajectory that it would follow in the original algorithm. However, this does not compromise optimality, because we uphold the same invariants: the path x-T-s_i serves as a witness that no shift towards x leads to a better shortcut, even if it is no longer diametral.

During Phase III, the path state has two components: the diametral path of type x-■ and the diametral path of type ■-y. We modify Phase III such that we ignore changes to the x-■-component of the path state when x-T-s_i has become diametral for some $i \in \{1, 2, \ldots, k\}$ until pq becomes useful for x, s_i. Likewise, we ignore changes to the ■-y-component when s_j-T-y has become diametral for some $j \in \{1, 2, \ldots, k\}$ until pq becomes useful for x, s_i.

Lemma 3.18. *For a geometric tree with n vertices, the modified Phase III has $O(n)$ events.*

Proof. In Phase III, the speed of p and q is determined by one type of diametral path for a diametral pair of type x-■ and by one type of diametral path for a diametral pair of type ■-y, as indicated in Table 3.2. Certain path state transitions are only possible when the shortcut shrinks, others only when the shortcut grows. Suppose, for instance, that the shortcut shrinks as we move from pq to $p'q'$, i.e., $|p'q'| < |pq|$. In this case, we cannot transition from {x-pq-●, x-T-●, ●-pq-y, ●-T-y} to {x-pq-▲, x-pq-●, x-T-●, ●-pq-y, ●-T-y} as the shortcut moves from pq to $p'q'$, since then $d_T(q,q') = d_T(p,p')$ any path of type x-pq-▲ shrinks by

$$d_{T+pq}(x,s_i) - d_{T+p'q'}(x,s_i) = d_T(p',p) + |pq| - |p'q'| - d_T(q',q) = |pq| - |p'q'|$$

whereas the diameter only shrinks by $\frac{1}{2}(|pq| - |p'q'|)$. Figure 3.25 illustrates the path state transitions during Phase III for the path states that do not contain x-T-▲ or ▲-T-y.

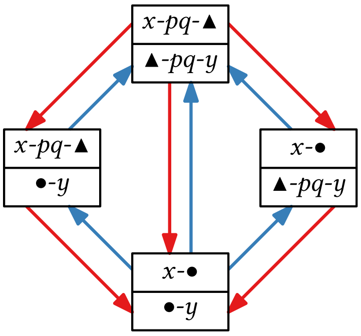

x-pq-$\blacktriangle$
$\blacktriangle$-pq-y
x-pq-$\blacktriangle$
$\bullet$-y
x-$\bullet$
$\blacktriangle$-pq-y
x-$\bullet$
$\bullet$-y

Figure 3.25: The path states encountered during Phase III, excluding the states containing x-T-$\blacktriangle$ or $\blacktriangle$-T-y and any transitions to these states. Red transitions may occur while the shortcut is shrinking; blue transitions may occur while the shortcut is growing.

Recall that the $\mathcal{B}$-sub-trees $S_1, S_2, \ldots, S_k$ are numbered in the order along the backbone from a to b. After x-T-s_i has become a diametral path of type x-T-$\blacktriangle$ in the modified Phase III, none of the paths x-T-s_j with $i < j$ will be registered as diametral paths. If x-T-s_j does become diametral after x-T-s_i has become diametral, then the shortcut is useless for x, s_i, since it is useless for x, s_j. Therefore, we do not register that x-T-s_j becomes diametral, because we are still waiting for the shortcut to become useful for x, s_i, since $d_T(x, s_i) = d_{T+pq}(s, x_i) \leq \operatorname{diam}(T + pq)$.

We argue that the modified Phase III visits the path states containing x-T-$\blacktriangle$ at most $k + n$ times. Figure 3.26 illustrates the possible changes in the x-$\blacksquare$-component of the path state during the modified Phase III. When the shortcut is growing, we only enter a path state containing x-T-$\blacktriangle$ when the path x-T-s_i that was most recently a diametral path of type x-T-$\blacktriangle$ becomes diametral again. In this case, the x-$\blacksquare$-component only changes away from x-T-$\blacktriangle$ after the shortcut has began to shrink again. Since the shortcut is growing at most n times during the modified Phase III, we register at most $O(n)$ path state changes of this kind. When the shortcut is shrinking, we only enter a path state containing x-T-$\blacktriangle$ when a path x-T-s_l becomes diametral that has not been diametral before. As argued above, this may occur at most k times, since $l < i$, where x-T-s_i was the most recent diametral path of type x-T-$\blacktriangle$. Likewise, we argue that the modified Phase III visits the path states containing $\blacktriangle$-T-y at most $k + n$ times.

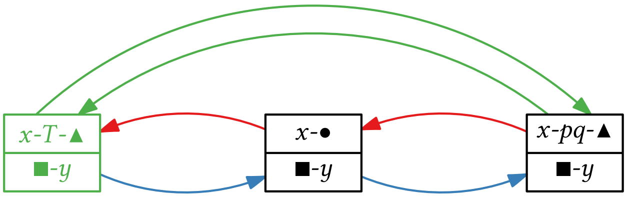


Figure 3.26: A simplified view on the path states encountered during Phase III. Only the changes in the path type for x-$\blacksquare$ are shown. Red transitions may occur while the shortcut is shrinking; blue transitions may occur while the shortcut is growing.

Once we have exhausted the at most $2k + 2n$ visits to path states containing x-T-$\blacktriangle$ or $\blacktriangle$-T-y, the remaining path state transitions form acyclic digraphs when the shortcut is shrinking and when the shortcut is growing, as illustrated in Figure 3.25. Since the shortcut changes at most $2n$ times between shrinking and growing, we register at most $2k + 4n$ events where the path state changes. Therefore, we process $O(n)$ events in total throughout the modified Phase III. □

We treat path state events during Phase III in the same fashion as in Phase I and II, except for the following two differences. First, the detection of diametral paths of type x-pq-▲ and ▲-pq-y changes, since x-pq-y is no longer diametral. Second, we need to detect when a diametral path appears that connects a diametral pair of type ■-■, since this marks the end of Phase III. The latter difference only concerns diametral paths of type ▲-T-▲ and ▲-pq-▲, since we already detect diametral paths of type ▲-pq-● and ▲-T-● when monitoring the candidates for diametral paths with an endpoint on the simple cycle $C(p, q)$ in the augmented tree $T + pq$.

x-pq-▲, ▲-pq-y There are only two cases in which a path x-pq-s_i, for some $i \in \{1, 2, \ldots, k\}$ becomes a diametral path of type x-pq-▲ during the modified Phase III.

In the first case, the path x-T-s_i is a diametral path of type x-T-▲ and the shortcut is about to become useful for x, s_i. We can detect this path state event in constant time per vertex event by comparing $d_T(a, r_i)$ with $d_T(a, \bar{p})$ to see when and if $\bar{p}$ passes through r_i.

In the second case, the shortcut is growing and the path x-pq-$\bar{p}$ is a diametral path of type x-pq-●. The point $\bar{p}$ is moving towards y when the shortcut is growing and s_i is a leaf of a secondary $\mathcal{B}$-sub-tree attached to the path from q to $\bar{p}$. This implies that we have $d_T(\bar{q}, r_i) = d_T(r_i, s_i)$ in this case. Therefore, we can detect this type of path state event by placing additional vertices at the points $\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_k$ along the backbone such that $\bar{p}_i$ is the point along the path from a to r_i with $d_T(\bar{p}_i, r_i) = d_T(r_i, s_i)$, if such a point exists. Placing these at most k additional vertices takes $O(n + k \log k)$ preprocessing time.

▲-T-▲ We may ignore diametral paths of type ▲-T-▲. Suppose that, during the modified Phase III, some path s_i-T-s_j, for $i, j \in \{1, 2, \ldots, k\}$, becomes a diametral path of type ▲-T-▲ at some position $\hat{p}\hat{q}$ of the shortcut. Even if we fail to register this path state event, we still report an optimal shortcut, because we report the shortcut that yields the smallest encountered continuous diameter including the continuous diameter of $T + \hat{p}\hat{q}$.

▲-pq-▲ We cannot ignore an event where a diametral path of type ▲-pq-▲ appears, because the length of these paths depends on pq. Moreover, we cannot afford to check whether a diametral path of type ▲-pq-▲ is about to appear when processing the other events. Therefore, we perform the detection of such events as a post-processing step instead.

It is sufficient to find the first position $\hat{p}\hat{q}$ where some path s_i-pq-s_j, for $i, j \in \{1, 2, \ldots, k\}$, becomes a diametral path of type ▲-pq-▲: If we shift outwards from $\hat{p}\hat{q}$, then s_i-pq-s_j will remain diametral while increasing in length. Thus, we proceed as follows.

First, we simulate the modified Phase III without attempting to detect if a diametral path of type ▲-pq-▲ appears. We record the sequence of edge pairs that we visit during this simulation. As argued in Lemma 3.18, this sequence contains $O(n)$ edge pairs. After the simulation, we perform a binary search for $\hat{p}\hat{q}$ in the sequence of visited edge pairs. The binary search for $\hat{p}\hat{q}$ takes $O(n \log n)$ time, since we can determine the largest path of type ▲-pq-▲ in $O(n)$ time for a fixed position of the shortcut, as shown in Lemma 3.19.

Lemma 3.19. *For every augmented tree $T + pq$ with n vertices, we can determine the length of the longest paths of type ▲-pq-▲ in $O(n)$ time.*

Proof. Every path of type ▲-pq-▲ has length $d_T(s_i, p) + |pq| + d_T(q, s_j)$ for some $i, j = 1, 2, \ldots, k$ with $i < j$ where pq is useful for s_i, s_j. This means s_i is a leaf of one of the secondary $\mathcal{B}$-sub-tees $S_{i_L}, S_{i_L+1}, \ldots, S_{i_H}$ attached to the path from p to $\bar{q}$, and s_j is a leaf of one of the secondary $\mathcal{B}$-sub-tees $S_{j_L}, S_{j_L+1}, \ldots, S_{j_H}$ attached to the path from $\bar{p}$ to q. We prove that the matrix M with

$$M_{j,i} = \begin{cases} d_T(s_i, p) + |pq| + d_T(q, s_j) & \text{, if } pq \text{ is useful for } s_i, s_j \\ 0 & \text{, otherwise} \end{cases} \quad \begin{array}{c} \text{for } i_L \le i \le i_H \\ \text{and } j_L \le j \le j_H \end{array} ,$$

is totally monotone. This means we have to show that $M_{j_1,i_1} < M_{j_1,i_2}$ implies $M_{j_2,i_1} < M_{j_2,i_2}$ for all indices i_1, i_2, j_1, j_2 with $i_L \le i_1 < i_2 \le i_R$ and $j_L \le j_1 < j_2 \le j_R$.

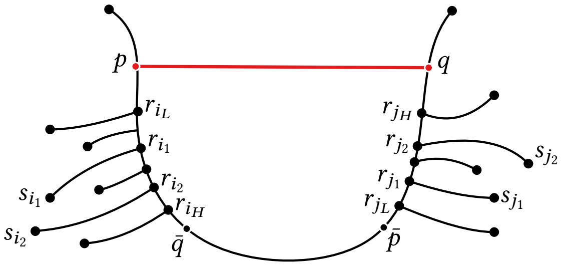


Figure 3.27: A sketch of the relative positions of the secondary $\mathcal{B}$-sub-trees with indices $i_L, i_1, i_2, i_H, j_L, j_1, j_2$, and j_H when determining the longest ▲-pq-▲ path.

Suppose we have $M_{j_1,i_1} < M_{j_1,i_2}$. Then $M_{j_1,i_2} > 0$, because all entries of M are non-negative. This means that pq is useful for s_{i_2}, s_{j_1}. Therefore, the shortcut pq is also useful for s_{i_1}, s_{j_1}, for s_{i_1}, s_{j_2}, and for s_{i_2}, s_{j_2}, due to the relative positions of r_{i_1} ,r_{i_2}, r_{j_1}, and r_{j_2}, as shown in Figure 3.27. Hence, $M_{j,i} = d_T(s_i, p) + |pq| + d_T(q, s_j)$ for $i \in \{i_1, i_2\}$ and $j \in \{j_1, j_2\}$. With this observation, $M_{j_1,i_1} < M_{j_1,i_2}$ implies $d_T(s_{i_1}, p) < d_T(s_{i_2}, p)$, since

$$d_T(s_{i_1}, p) + |pq| + d_T(q, s_{j_1}) = M_{j_1,i_1} < M_{j_1,i_2} = d_T(s_{i_2}, p) + |pq| + d_T(q, s_{j_1}) \ ,$$

which, in turn, implies $M_{j_2,i_1} < M_{j_2,i_2}$, because

$$M_{j_2,i_1} = d_T(s_{i_1}, p) + |pq| + d_T(q, s_{j_2}) < d_T(s_{i_2}, p) + |pq| + d_T(q, s_{j_2}) = M_{j_2,i_2} \ .$$

Therefore, the matrix M is totally monotone. We can access any entry $M_{j,i}$ of M in constant time after $O(n)$ pre-processing due to the following. We determine $d_T(s_1, r_1), d_T(s_2, r_2), \ldots, d_T(s_k, r_k)$ as well as $d_T(a, r_1), d_T(a, r_2), \ldots, d_T(a, r_k)$ in advance. The shortcut pq is useful for s_i, s_j precisely when $d_T(r_i, p) + |pq| + d_T(q, r_j) < d_T(r_i, r_j)$, which we can check in constant time with our preparation. Computing $d_T(s_i, p) + |pq| + d_T(q, s_j)$ also takes constant time.

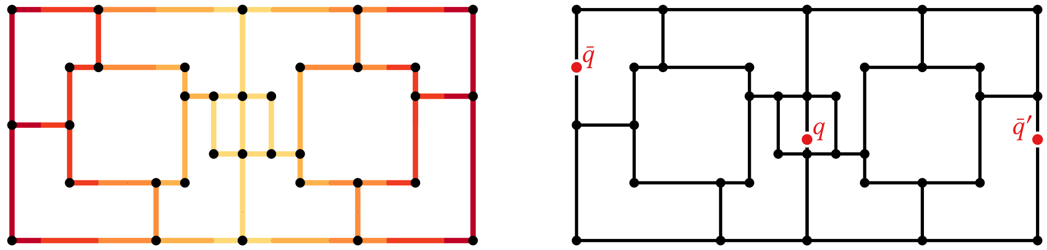
Therefore, we can determine a largest entry in M—and, thus, a longest path of type ▲-pq-▲—in $O(n)$ time using the SMAWK Algorithm [2] without constructing M explicitly. ☐

In conclusion, we can discretize all phases of the continuous algorithm—with some modifications that do not impact optimality—with $O(n)$ events that we can process in $O(n \log n)$ total time, followed by a post-processing step that takes $O(n \log n)$ time.

Theorem 3.20. *For a geometric tree T with n vertices, it takes $O(n \log n)$ time to determine a shortcut pq for T that minimizes the continuous diameter of the augmented tree $T + pq$.* ☐

4 Data Structures for Farthest-Point Queries

We aim to design efficient data structures that answer the following types of queries for a fixed network G. Given a point q on the network G, what is the farthest distance from q in G? What are the farthest points from q in G? We refer to the former as *farthest-distance query* and to the latter as *farthest-point query*. Figure 4.1 illustrates these queries in a geometric network.



(a) A heat map of the farthest distance.

(b) A farthest-point query with its answer.

Figure 4.1: Farthest-distance queries and farthest-point queries in a geometric network G. In Subfigure (a), the edges of G are shaded depending on the farthest distance with the values grouped in five intervals; brighter shades indicate low values and darker shades indicate higher values. The farthest distance is low at the center and high at its fringe. Subfigure (b) illustrates a farthest-distance query from a point q whose farthest points in G are the points $\bar{q}$ and $\bar{q}'$ that are points along edges, not vertices.

A query point q is represented by the edge e containing q together with the value $\lambda \in [0, 1]$ that specifies the relative position of q along e with respect to the endpoints u and v of e, i.e., $d_e(u, p) = \lambda \cdot d_e(u, v)$. The farthest points from q are represented in the same fashion. The *farthest distance* from $q \in G$ is denoted by $\bar{d}_G(q)$, i.e., $\bar{d}_G(q) := \max_{p \in G} d_G(p, q)$, and a point $p \in G$ is a *farthest point* from q in the network G when $\bar{d}_G(q) = d_G(p, q)$.

4.1 Preliminaries

Let G be a network with n vertices and m edges. If we perform no preprocessing, then we can answer farthest-distance queries and farthest point queries in G in $O(m + n \log n)$ time using Dijkstra's Algorithm with Fibonacci heaps [26]: For a query $q \in G$, we compute a shortest path tree T_q that is rooted at q. For each edge st that is not part of T_q, we subdivide st into two sub-edges sx and xt such that for all points $p \in sx$ a shortest path to q passes through s and for all points $p \in xt$ a shortest path to q passes through t. Orienting the edges of the resulting subdivided network towards q yields an *extended shortest path tree* [48] rooted at q. Notice that—despite of the name—an extended shortest path tree is a directed acyclic graph and not necessarily a tree. Every farthest point from q in G is a source of any extended shortest path tree rooted at q. Figure 4.2 illustrates an example of an extended shortest path tree.

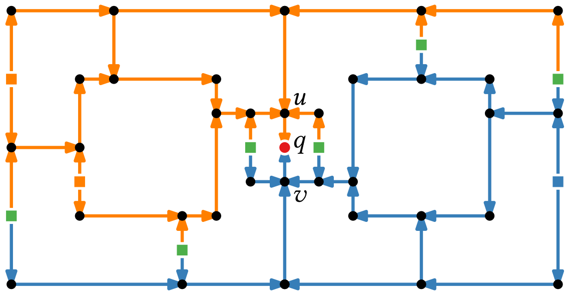
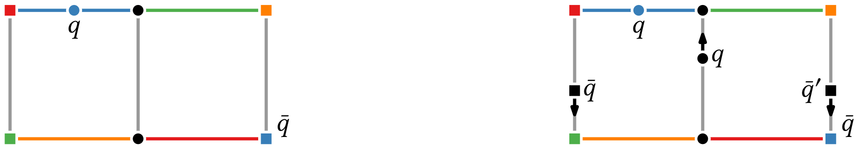


Figure 4.2: An extended shortest path tree E_q that is rooted at the point q in a geometric network. The edges of E_q along shortest paths to q via u (and the sub-edge uq) are coloured orange and the shortest paths to q via v (and the sub-edge qv) are coloured blue. The monochromatic sources (orange squares and blue squares) of E_q are candidates for stationary farthest points from q and the bichromatic sources (green squares) that are incident to edges of different colour are candidates for moving farthest points.

Since all points along a network G are candidates for farthest points, we observe two types of changes in the set of farthest points when moving the query point along an edge. As illustrated in Figure 4.3, there are farthest points that remain stationary when moving the query point, and there are farthest points that move with the query point. Hence, the set of points that are farthest from some point on the network may be uncountable infinite. This means that there may be uncountably infinitely many regions in the farthest-point Voronoi diagram that is defined with respect to the metric space formed by all points along a network and their network distance, and where all points along the network are considered to be sites. We obtain a finite representation of this infinite diagram when by subdividing the network depending on which edges contain farthest points [12]; we call this finite representation the *network farthest-point diagram.*



(a) Stationary farthest point.

(b) Moving farthest points.

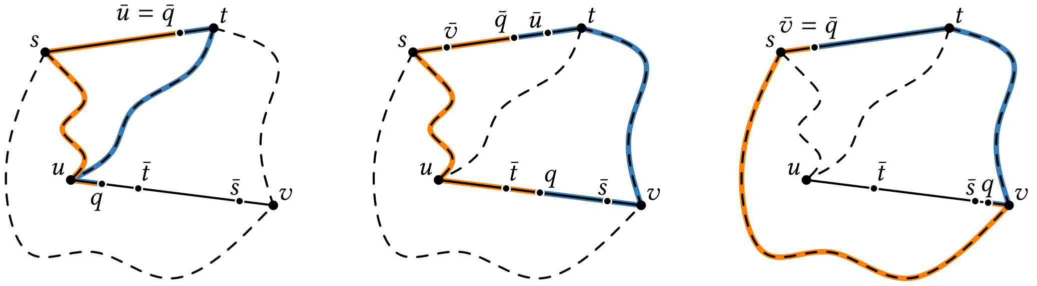
Figure 4.3: A network with (a) stationary and (b) moving farthest points. Each point on the blue edge has the blue square as a stationary farthest point. Each point p on the vertical middle edge has two farthest points $\bar{p}$ and $\bar{p}'$ that move downwards as p moves upwards. No two points on the middle edge have the same farthest points.

For a network G with n vertices and m edges, the network farthest-point diagram of G has size $O(m^2)$, construction time $O(m^2 \log n)$, and supports $O(\log n)$-time farthest-distance queries and

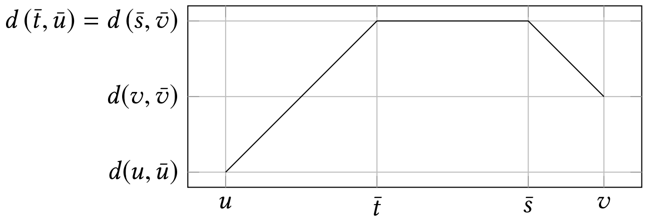
$O(k + \log n)$-time farthest-point queries in G, where k is the number of reported farthest points. The purpose of this chapter is to develop more efficient data structures for certain networks.

4.1.1 Previous Work

This chapter builds upon previous results on data structures supporting farthest-distance queries and farthest-point queries in networks [12, 13, 27]. It is instructive to review these results here, as we combine data structures for simpler networks to support queries in more complex networks.



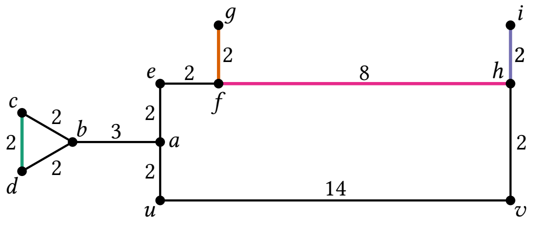
(a) A query facing u. (b) A query facing both ways. (c) A query facing v.



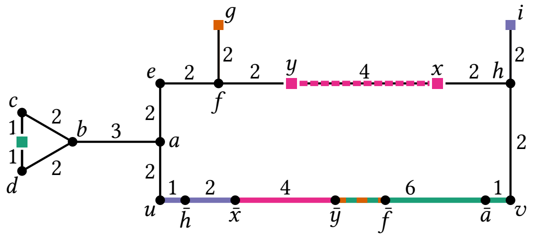
(d) The farthest distance from $q \in uv$ to any point on st.

Figure 4.4: A derivation of the farthest distance from an edge uv of a network N to any point on another edge st. Let $\bar{t}$ and $\bar{s}$ be the farthest points in N from t and s along uv, and let $\bar{u}$ and $\bar{v}$ be the farthest points in N from u and v along st. If the query q lies between u and $\bar{t}$ with $q \neq \bar{t}$, as in Subfigure (a), then the farthest point from q on st is $\bar{u}$. If the query q moves from $\bar{t}$ to $\bar{s}$ along uv, as in Subfigure (b), then the farthest point from q on st moves from $\bar{u}$ to $\bar{v}$. If the query q lies between $\bar{s}$ and v with $q \neq \bar{v}$, as in Subfigure (c), then the farthest point from q on st is $\bar{v}$. For a point q on edge uv, the function $\bar{d}_{st}(q) = \max_{p \in st} d_N(p, q)$ has the shape depicted in Subfigure (d).

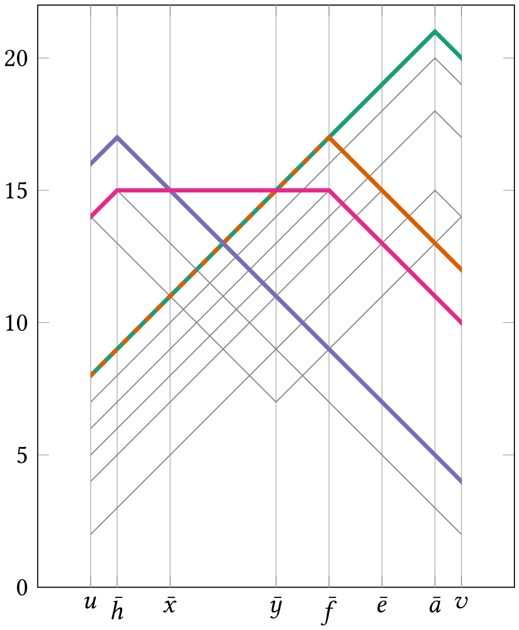
General Networks [12, 27] We consider an arbitrary network N. For each pair of edges uv and st of N, we determine the farthest distance from points on uv to any point along st as a function of the position of the query along uv, i.e., we compute $\bar{d}_{st}(q) = \max_{p \in st} d_N(p, q)$ for each $q \in uv$, as illustrated in Figure 4.4. Then, we compute the upper envelope of the functions indicating farthest distances from uv to support farthest distance queries from uv, i.e., $\bar{d}_N(q) = \max_{st \in E} \bar{d}_{st}$ for $q \in uv$, where E is the set of edges of N. An example is illustrated in Figure 4.5. We repeat this for each edge and construct a data structure that reports the functions determining the upper envelope, and, thereby, the edges containing farthest points. For a network with n vertices and m edges, this leads to a data structure with size $\Theta(m^2)$ and construction time $O(m^2 \log n)$ that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries, where k is the number of reported farthest points. The bound on the size of the data structure is tight, i.e., for every value of m, we can construct a network with m edges, where the set of the edges containing farthest points changes $\Omega(m^2)$ times as we traverse all edges of the network. Notice that this is a lower bound for the approach, not the problem itself: there may exist a data structure for farthest-point queries in general networks with size $o(m^2)$.



(a) An network N with some highlighted edges.



(b) The subdivision of the edge uv depending on the location of the farthest points in N.



(c) The edge-wise farthest distance functions for uv.

Figure 4.5: The computation of the farthest distance function from an edge uv in a network N. In Subfigure (a), the edges of N that contain farthest points from some point along uv are highlighted in the colours that match the corresponding edge-wise farthest distance functions in Subfigure (c). Based to the upper envelope of these functions, we subdivide N depending on the location of the farthest points from queries along uv, as illustrated in Subfigure (b). The colours of the sub-edges of uv in Subfigure (b) indicates that a square of matching colour is a stationary farthest point or that there is a moving farthest point along the sub-edge xy. For each $\bullet \in \{h, x, y, f, e, a\}$, the point $\bar{\bullet}$ denotes the farthest point from $\bullet$ along uv with respect to the cycle in N.

Cycle Networks [13, 27] Let C be a cycle network with n edges. The length of C is denoted as $|C|$. For any point $q \in C$ the farthest point from q in C is the antipodal $\bar{q}$ of q and $\bar{d}_C(q) = d_C(q, \bar{q}) = |C|/2$. Hence, answering farthest-distance queries from a cycle takes constant time, once we have calculated $|C|$. To locate farthest points, we subdivide C at the antipodal $\bar{v}$ of every vertex v of C and introduce pointers between pairs of antipodal vertices, as illustrated in Figure 4.6. This allows us to perform a binary search for the antipodal point for any query point. Therefore, for every cycle C, there exists a data structure with $O(n)$ size and $O(n)$ construction time that supports $O(1)$-time farthest-distance queries and $O(\log n)$-time farthest-point queries on C. Notice that we cannot improve the query time for farthest-point queries, since this would also mean an improvement to predecessor search in a sorted array, as illustrated in Figure 4.7. However, if we are allowed to subdivide C with $O(n)$ additional vertices (the antipodals of the vertices), and if we are allowed to specify queries with respect to the subdivided edges instead of the original edges of C, then we can answer farthest-point queries in constant time by storing the edge containing the antipodal with each edge in the sub-divided cycle.

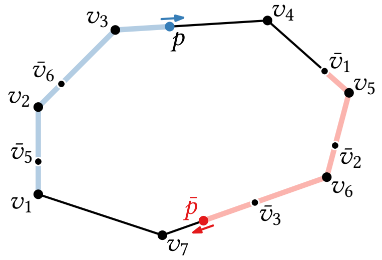


Figure 4.6: The construction for farthest-point queries on a cycle C. For some vertex v_1 of C, we locate the antipodal point $\bar{v}_1$ of v_1. Starting from v_1 and $\bar{v}_1$, we sweep a point p and its antipodal $\bar{p}$ along the cycle and introduce a new vertex at p or at $\bar{p}$ when the other point reaches a vertex. We store a pointer from each vertex to its antipodal and vice versa to locate the edge containing the farthest point from a query point.

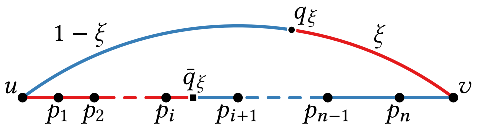
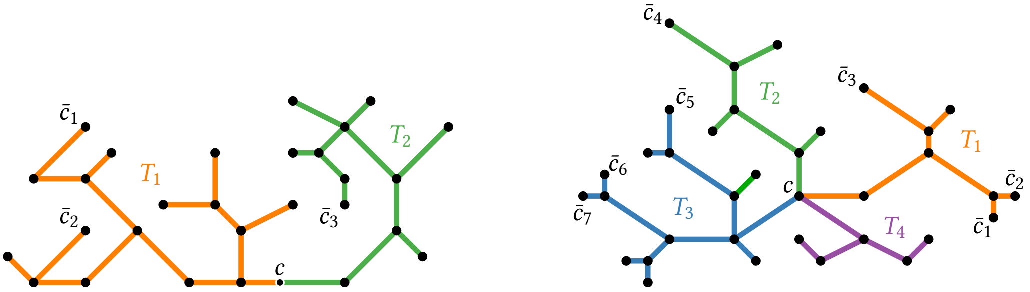


Figure 4.7: The reduction from predecessor search in sorted arrays to farthest-point queries on cycles. Suppose we are given a sorted array A with entries $a_1, a_2, \ldots, a_n$ where $0 < a_1 < a_2 < \cdots < a_n < 1$. We construct the depicted cycle C: we begin with two edges e and e' of weighted length one that connect two vertices u and v. We subdivide the edge e' with points $p_1, p_2, \ldots, p_n$ such that $d(u, p_i) = a_i$, for each $i = 1, 2, \ldots, n$. We answer a query for the predecessor of $\xi \in (0, 1)$ in A with a farthest-point query in C from the point $q_\xi \in e$ with $d(q_\xi, v) = \xi$. The farthest point $\bar{q}_\xi$ from q_ξ in C lies on the edge connecting p_i and p_{i+1} if and only if a_i is the predecessor of ξ in A.

Tree Networks [13, 27] Let T be a tree network with n vertices. Every tree has exactly one pivotal point where the set of farthest points changes. This pivotal point is the *absolute center* of T, i.e., the point $c \in T$ that minimizes the farthest distance to the leaves of T. The absolute center c of T is the midpoint of every diametral path of T, i.e., $\bar{d}_T(c) = \mathrm{diam}(T)/2$. As illustrated in Figure 4.8, we split T at c into sub-trees $T_1, T_2, \ldots, T_l$, where each T_i is a maximal sub-tree of T consisting of points that can reach each other via a path that does not contain c in its interior. For every point $q \in T_i$ with $q \neq c$, the farthest points from q in T are the farthest leaves from c in the sub-trees T_j with $j \neq i$. The farthest distance from any point $q \in T_i$ is the distance between q and c plus the distance from c to the farthest leaves from c in $T \setminus T_i$, i.e., $\bar{d}_T(q) = d_T(q, c) + \mathrm{diam}(T)/2$. It takes $O(n)$ time to compute the absolute center c, the diameter of T, and the farthest leaves from c in each sub-tree $T_1, T_2, \ldots, T_l$. Therefore, for every tree T, there exists a data structure with size and construction time $O(n)$ that supports $O(1)$-time farthest-distance queries and $O(k)$-time farthest-point queries in T, where k is the number of reported farthest points.

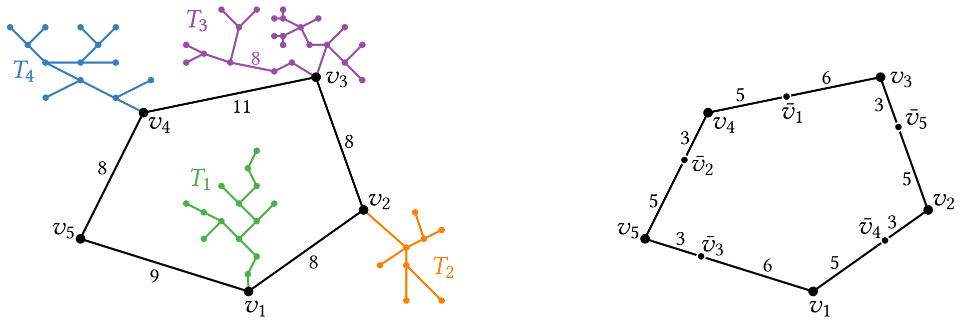


(a) The absolute center lies on an edge.

(b) The absolute center lies on a vertex.

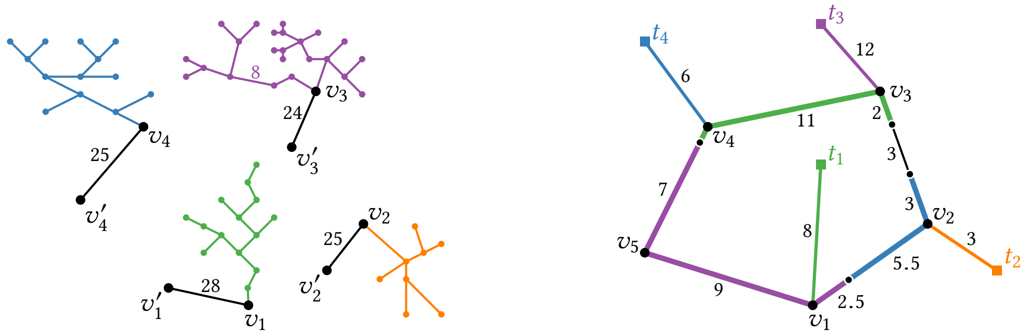
Figure 4.8: The decomposion of two geometric trees into sub-trees with the same farthest points. When the absolute center c of a tree T falls on an edge, as depicted in Subfigure (a), then we split T into two sub-trees. When the absolute center c of a tree T falls onto a vertex of degree l, as shown in Subfigure (b), then we split T into l sub-trees. The farthest points for $q \in T_i$ with $q \neq c$ are the farthest leaves from c in $T \setminus T_i$.

Uni-Cycle Networks [13, 27] A *uni-cyclic network* is a network with exactly one cycle. This means that every uni-cyclic network U consists of a cycle C with trees $T_1, T_2, \ldots, T_l$ that are attached to C at vertices $v_1, v_2, \ldots, v_l$. As illustrated in Figure 4.9, we support queries in a uni-cyclic network by combining the data structure for trees, the data structure for cycles, and a new data structure that identifies the trees that contain farthest points from a query point along the cycle. The following observations show that it takes $O(n)$ time to subdivide the cycle C depending on which trees among $T_1, T_2, \ldots, T_l$, if any, contain farthest points. We call a tree T_i *relevant* if some point on C has a farthest point in T_i. A tree T_i is relevant if and only if the antipodal point $\bar{v}_i$ of v_i has a farthest point in T_i. The points along C that have farthest points in T_i form a path along C. These paths appear in the same circular order as their corresponding relevant trees appear along C.



(a) A uni-cyclic network.

(b) Data structure for the cycle.



(c) The perspectives from the trees.

(d) The perspective from the cycle.

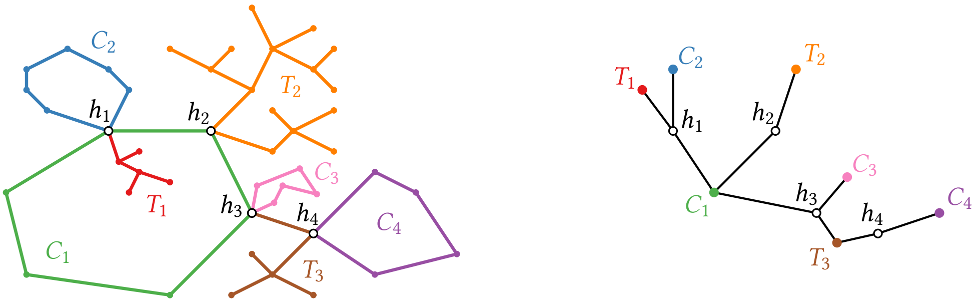
Figure 4.9: A uni-cyclic network together with the perspectives from its trees and from its cycle. The edge weights are omitted for edges of weight one. In the perspective from the cycle, the colouring of the cycle indicates which tree, if any, contains farthest points.

For uni-cyclic networks, we introduce the idea of the perspective from a sub-network on a network. The *perspective* from tree T_i on U is the tree that consists of T with a new edge e from v_i to a new vertex v_i' whose weighted length w_e is the farthest distance from v_i to any point in U outside of T_i, i.e., $w_e = \bar{d}_{U \setminus T_i}(v_i)$. The *perspective* from the cycle C onto U consists of C where each sub-tree T_i is replaced by a pendant edge e_i from v_i to a new vertex t_i whose weighted length w_{e_i} is the farthest distance from v_i to any point in T_i, i.e., $w_{e_i} = \bar{d}_{T_i}(v_i)$. As illustrated in Figures 4.9c and 4.9d, the perspectives from the trees and cycles preserve the distance information from the original network and allow us to split queries into sub-queries that we can handle with known data structures.

For a uni-cyclic network U with n vertices, there exists a data structure with $O(n)$ size and construction time that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in U, where k is the number of reported farthest points.

Cactus Networks [13, 27] A cactus network is a network where no two simple cycles share an edge. This means that every cactus network N consists of cycles $C_1, C_2, \ldots, C_\eta$ that are connected by trees $T_1, T_2, \ldots, T_\nu$, as illustrated in Figure 4.10. To support queries in cactus networks, we expand on the idea of perspectives. The perspective from a cycle C_i

on a cactus network N is a uni-cyclic network and the perspective from a tree T_j on N is a tree. Since we already know how to support queries in trees and uni-cyclic networks, the challenge for cactus networks lies in constructing the perspectives from the trees and the cycles of a cactus and in connecting their corresponding data structures.



(a) A cactus network N.

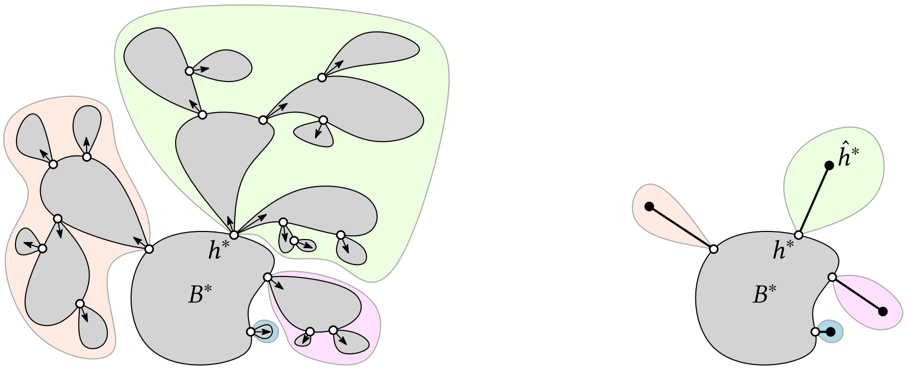
(b) The tree structure of N.

Figure 4.10: A cactus network with its tree structure.

We describe the construction of the perspectives from the bi-connected components of a network as illustrated in Figure 4.11. We construct the perspectives from trees and from cycles on a cactus network in the same manner. We start with some bi-connected component B^* and recursively determine the farthest distance on paths leading away from the bi-connected component B^* with a breadth-first search in the tree structure of the network. Then, we propagate the information gathered at B^* to the other bi-connected components to build the remaining perspectives. As illustrated in Figure 4.12, we introduce shortcuts in the tree structure to support efficient farthest point queries. These shortcuts point to the next bi-connected component in the tree structure that contains a farthest point, or to the next bi-connected component where two paths to farthest points diverge.

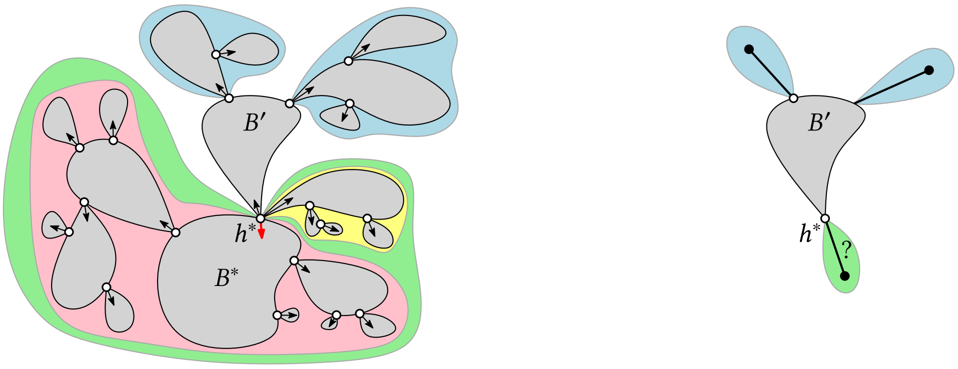
For every cactus N with n vertices, there exists a data structure with $O(n)$ size and construction time that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in N, where k is the number of reported farthest points.

Tree-Like Networks [13, 27] The strategy for cactus networks generalizes as follows. Consider a network N with n vertices and b bi-connected components $B_1, B_2, \dots, B_b$ of sizes $n_1, n_2, \dots, n_b$, respectively. Suppose we have a data structure with size $S_i(n_i)$ and construction time $T_i(n_i)$ for the perspective from B_i on N, for each $i = 1, 2, \dots, b$. Suppose that this data structure supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries from points on B_i and that it supports $O(1)$-time farthest-distance queries and $O(k)$-time farthest-point queries from dummy vertices in the perspective from B_i. Then, there exists a data structure with size $O\left(n + \sum_{i=1}^{b} S_i(n_i)\right)$ and construction time $O\left(n + \sum_{i=1}^{b} T_i(n_i)\right)$ that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in N, where k is the number of reported farthest points.



(a) A bi-connected component B^* in a network N.

(b) The perspective from B^* on N.

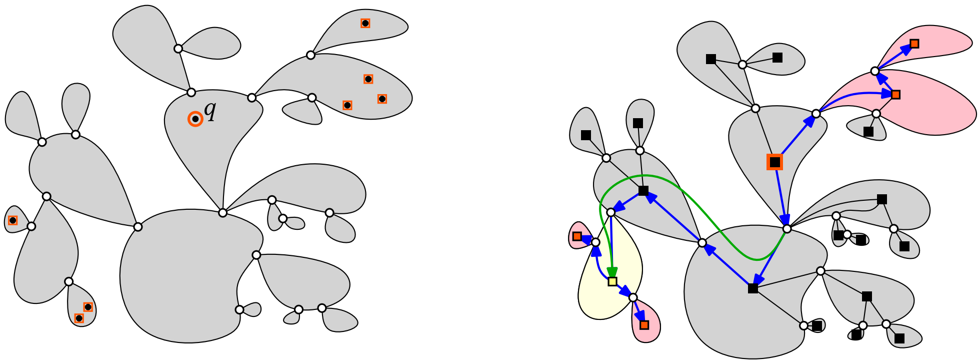


(c) A bi-connected component B' neighbouring B^*.

(d) The perspective from B' on N.

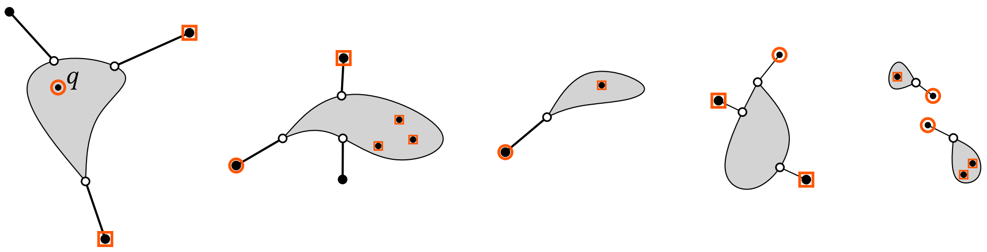
Figure 4.11: The construction of the perspectives from the bi-connected components (gray shapes) on a network N. We begin with some bi-connected component B^*. For each connected component X (coloured shapes) of $N \setminus B^*$ with at least one edge, we determine the farthest distance $\bar{d}_X(h^*)$ in X from the cut vertex h^* (empty circles) that connects B^* with the bi-connected components in X. We compute the longest paths leading away from B^* in X, and thus $\bar{d}_X(h^*)$, with a breadth-first search in the tree structure (black arrows). Using the perspective from B^*, we construct the perspective of each neighbouring bi-connected component B' of B^*. We already know all but one of the required distances for the construction of the perspective from B' on N from the construction for B^*. We obtain the missing distance (red arrow) with a query in the perspective from B^* on N from the artificial vertex $\hat{h}^*$ that represents the connected component X in $N \setminus B^*$ that contains the bi-connected component B'.

The term *tree-like*—which is not formally defined—expresses that we consider a network to be more like a tree when the network decomposes into a large number of small bi-connected components. If a network decomposes into bi-connected components of constant size, then the above leads to a data structure with size and construction time $O(n)$. On the other hand, if the network decomposes into bi-connected components of linear size, then we gain little from this strategy. Despite lacking a formal definition, we would consider the former network to be more tree-like than the latter.



(a) A query q from a bi-connected component of a network N.

(b) The paths to farthest points from q in N.



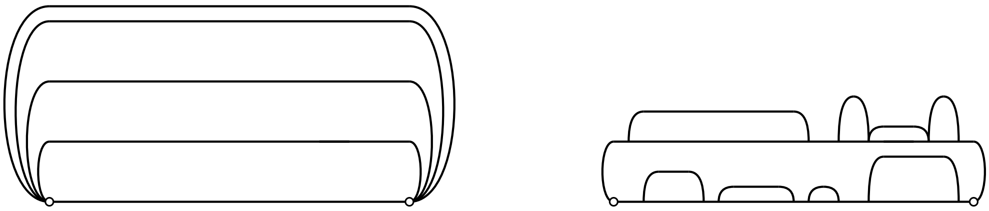
(c) The perspectives that are visited when answering the query from q in N.

Figure 4.12: A query q for the farthest points (squares) from q in a network N with respect to the tree structure of N. In Subfigure (b), the bi-connected components that contain farthest points from q are red and a bi-connected component where paths to farthest points diverge are yellow. We follow a shortcut (green) to skip some bi-connected components along the paths (blue) to the red or yellow bi-connected components. Subfigure (c) illustrates the queries in the perspectives from all bi-connected components on N that are accessed when processing the query from q.

The connection between the previous works and this work is as follows. The strategy for tree-like networks applies to any network, yet its success depends on how well we can handle the larger bi-connected components. We aim to expand the usefulness of the strategy for tree-like networks with more sophisticated bi-connected components than trees and cycles. One type of networks that appears to be particularly hard for this problem are grid networks, since small changes in the weight of the grid edges can cause vast changes in the extended shortest path tree from a query point. In order to avoid networks with large grid minors, we consider networks of low treewidth, starting with networks of treewidth two, i.e., series-parallel networks.

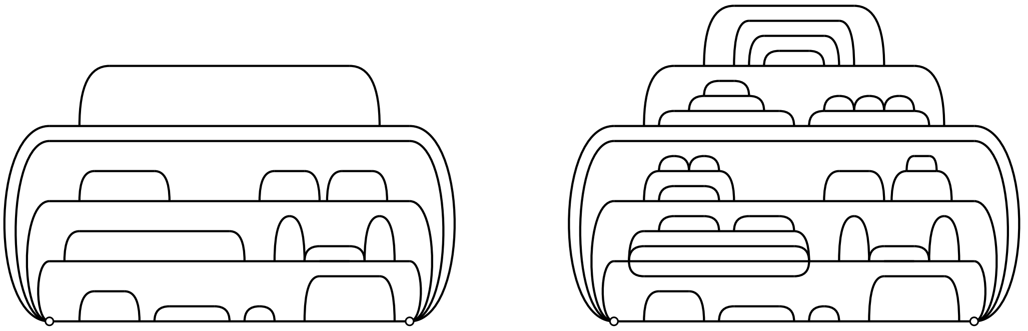
4.1.2 Structure and Results

We obtain the data structure for series-parallel networks by studying intermediate types of networks. We begin with networks reflecting parallel structure (parallel-path networks) and serial structure (bead-chain networks). Combining them, we support queries on flat two-terminal series-parallel networks (abacus). We then decompose two-terminal series-parallel networks into a tree of nested networks and combine their associated data structures. We tackle series-parallel networks by first building separate data structures for each of its bi-connected components, which are two-terminal series-parallel by definition. Finally, we connect the data structures for the bi-connected components exploiting that the bi-connected components are connected in a tree-like fashion. Table 4.1 summarizes the proposed data structures and compares them to previous results. Figure 4.13 illustrates the types of networks that we study in this work.



(a) A parallel-path network.

(b) A bead-chain network.



(c) An abacus network.

(d) A two-terminal series-parallel network.

Figure 4.13: The types of two-terminal series-parallel networks studied in this work.

Parallel-Path Networks A parallel-path network N is obtained by applying the parallel operation λ times to a single edge and subdividing the resulting network with series operations. As illustrated in Figure 4.13a, every parallel-path network N consists of λ paths that only meet at the two terminals u and v of N and that are otherwise disjoint.

We support queries in parallel-path networks by distinguishing queries where all farthest points are reached through the terminal u (queries facing u), all farthest points are reached through the terminal v (queries facing v), or neither (queries facing both ways). Furthermore, we distinguish candidates for farthest points on the u-v-path containing the query (inward queries) and candidates on other u-v-paths (outward queries).

Type	Farthest-Point Query	Size	Construction Time	Reference
General	$\Theta(k + \log n)$	$O(m^2)$	$O(m^2 \log n)$	[12, 27]
Tree	$\Theta(k)$	$\Theta(n)$	$\Theta(n)$	[13, 27]
Cycle	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$	[13, 27]
Uni-Cyclic	$\Theta(k + \log n)$	$\Theta(n)$	$\Theta(n)$	[13, 27]
Cactus	$\Theta(k + \log n)$	$\Theta(n)$	$\Theta(n)$	[13, 27]
Parallel-Path	$\Theta(k + \log n)$	$\Theta(n)$	$\Theta(n)$	[28]
Bead-Chain	$\Theta(k + \log n)$	$\Theta(n)$	$\Theta(n)$	[28]
Abacus	$\Theta(k + \log n)$	$\Theta(n)$	$O(n \log \lambda)$	[28]
Two-Terminal	$\Theta(k + \log n)$	$O(\tau n)$	$O(\tau n \log \lambda)$	
Series-Parallel	$\Theta(k + \log n)$	$O(\tau n)$	$O(\tau n \log \lambda)$	

Table 4.1: The traits of our data structures for queries in different types of networks, with n vertices, m edges, k reported farthest points, nesting number τ, and parallelism λ.

For a parallel-path network with n vertices, we obtain a data structure with size and construction time $O(n)$ that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries, where k is the number of reported farthest points.

Bead-Chains A bead-chain network is obtained from a cycle C by applying one parallel operation to σ edges of C and subdividing the resulting network with series operations. As illustrated in Figure 4.13b, a bead-chain network consists of a main cycle C with paths that are attached to C in a non-overlapping fashion; we refer to these paths as *arcs*.

We support queries in bead-chain networks by distinguishing whether the query originates from the main cycle or from an arc. For queries from the main cycle, we determine the distance to the farthest point on the cycle itself and on each arc as a function of the position of the query. Then, we compute the upper envelope of these functions to describe the farthest distance from points along the cycle. Due to the shape of these functions, this takes linear time. We support farthest-point queries from the cycle using an interval stabbing data structure that reports the arcs whose functions match the upper envelope. For queries from the arcs, we project the query point onto the cycle and reconstruct the answer for the original query on the arc from the projected query on the cycle.

For a bead-chain network B with n vertices, this yields a data structure with size and construction time $O(n)$ that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in B, where k is the number of reported farthest points. This data structure generalizes the data structure for queries in a uni-cyclic network.

Abacus Networks An abacus network is obtained from a parallel-path network N by applying one parallel operation to σ edges of N and subdividing the resulting network with series operations. As illustrated in Figure 4.13c, an abacus consists of a number of parallel paths $P_1, P_2 \ldots, P_\lambda$ that connect the terminals, and each of these parallel paths has additional paths attached to it in a non-overlapping fashion; we refer to these attached paths as *arcs*. Furthermore, we call each parallel path P_i with its arcs a *bead-string* and denote it by B_i.

We support queries in abacus networks by building on the techniques for parallel-path networks and bead-chain networks. We distinguish whether the query reaches its farthest points through the terminal u (queries facing u), through the terminal v (queries faching v), or neither (queries facing both ways). We translate queries from the arcs to queries on the parallel-path network similar to how we translate queries from the arcs of a bead-chain to its main cycle. Abacus networks require a number of new approaches as well: we split farthest-point queries into queries for the farthest points on the bead-string containing the query (inward queries) and queries for the farthest points outside the bead-string containing the query (outward queries). We support inward queries by completing each bead-string B_i to a bead-chain with an additional edge that represents the shortest path connecting the terminals outside of the bead-string B_i. We support outward queries by translating them to a virtual edge from where we compute the functions that describe the farthest distance to the arcs of each of the λ bead-strings. We recover the answer to the outward farthest-point query from the first layer (i.e., the upper envelope) or the second layer (i.e., the functions directly below the upper envelope) of these functions.

For an abacus A with n vertices, this leads to a data structure with $O(n)$ size and $O(n \log \lambda)$ construction time that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in A, where k is the number of reported farthest points.

Two-Terminal We decompose a bi-connected two-terminal series-parallel network N into a hierarchy $\mathcal{H}$ of nested abacus networks. Intuitively, the hierarchy $\mathcal{H}$ is formed as follows. We generate N starting with an abacus A that becomes the root node of $\mathcal{H}$. We select a cycle C in A with exactly two degree-three vertices a and b in A. We replace C with an abacus with terminals a and b that contains C and repeat this process on the resulting network. Whenever a cycle is replaced with an abacus, the new abacus becomes a child node in $\mathcal{H}$ of the node containing the cycle. The *nesting number* τ of N is the size of $\mathcal{H}$, i.e., τ is the number of times we replace a cycle with an abacus network to obtain N.

We support queries in two-terminal series-parallel networks by generalizing the approach for abacus networks. For each node of the nesting hierarchy, we construct a data structure that aggregates the farthest-distance information from other parts of the network in the same fashion as we aggregate the farthest distance information from the arcs of an abacus. For a two-terminal series-parallel network N with n vertices, parallelism λ, and nesting number τ, this leads to a data structure with size $O(\tau n)$ and construction time $O(\tau n \log \lambda)$ that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in N, where k is the number of reported farthest points.

Series-Parallel We apply the strategy for tree-like networks to series-parallel networks, i.e., we decompose a series-parallel network into its bi-connected components and build data structures for the bi-connected components and for the perspectives from the bi-connected components on the network. The bi-connected components of a series-parallel network are, by definition, two-terminal series-parallel; the perspective from each bi-connected component consists of a two-terminal series-parallel with attached pendant edges. Even though these perspectives are not two-terminal series-parallel, we argue that we can still use our data structure for two-terminal series parallel networks for them, because we can interpret the pendant edges as arcs whose endpoints coincide.

Let N be a series-parallel network with n vertices, and b bi-connected components of size $n_1, n_2, \ldots, n_b$, nesting numbers $\tau_1, \tau_2, \ldots, \tau_b$, and parallelisms $\lambda_1, \lambda_2, \ldots, \lambda_b$. Moreover, let $\tau = \max_{i=1}^{b} \tau_i$ be the largest nesting number and let $\lambda = \max_{i=1}^{b} \lambda_i$ be the largest parallelism in N. By applying the strategy for tree-like networks, we obtain a data structure with size $O\left(n + \sum_{i=1}^{b} \tau_i n_i\right) = O(\tau n)$ and construction time $O\left(n + \sum_{i=1}^{b} \tau_i n_i \log \lambda_i\right) = O(\tau n \log \lambda)$ that supports $O(\log n)$ time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in N, where k is the number of reported farthest points.

4.2 Parallel-Path Networks

A *parallel-path network* N consists of λ edge disjoint paths $P_1, P_2, \ldots, P_\lambda$ that connect two vertices u and v, as illustrated in Figure 4.14. Parallel-path networks are series-parallel networks that are generated from an edge uv using λ parallel operations followed by a sequence of series operations to further subdivide the paths. Without loss of generality, let $|P_1| \leq |P_2| \leq \cdots \leq |P_\lambda|$ be the weighted lengths of $P_1, P_2, \ldots, P_\lambda$, respectively.

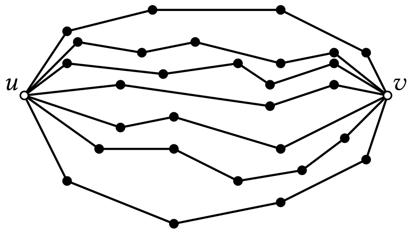


Figure 4.14: A parallel-path network.

For a query point $q \in N$, we consider an *extended shortest path tree* [48] rooted at q, i.e., a directed acyclic graph that results from directing all edges towards q after splitting each edge st that is not in a shortest path tree for q into two sub-edges sx and xt, where all points on sx reach q through s and all points on xt reach q through t. As illustrated in Figure 4.15, either all shortest paths from q reach v via u (i.e., q is *facing* u), or all shortest paths from q reach u via v (i.e., q is *facing* v), or neither (i.e., q is *facing both ways*). We distinguish the three cases using the following notation. Let $\bar{x}_i$ denote the farthest point from $x \in \{u, v\}$ among the points of path P_i, i.e., $\bar{x}_i$ is the point on P_i such that $d_N(x, \bar{x}_i) = \max_{y \in P_i} d_N(x, y)$. Together with Figure 4.15, the next lemma characterizes when the query point q is *facing u*, *facing both ways*, or *facing v*.

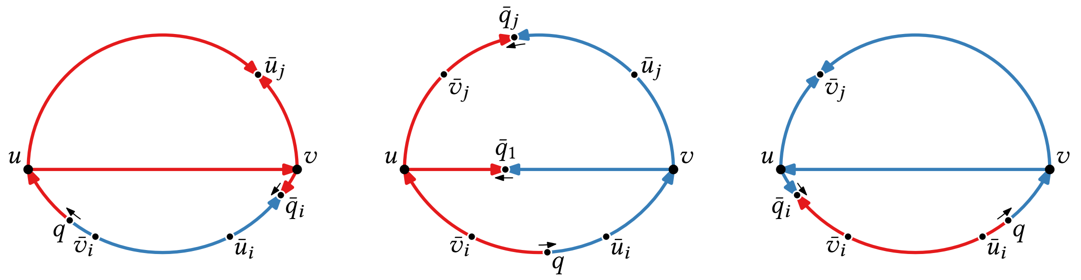
Lemma 4.1. *Let N be a parallel-path network with terminals u and v that are connected by the edge disjoint paths $P_1, P_2, \ldots, P_\lambda$. Consider a query point $q \in P_i$ for any $i = 1, 2, \ldots, \lambda$.*

(i) The query q is facing u if and only if q lies on the sub-path from u to $\bar{v}_i$ with $q \neq \bar{v}_i$,

(ii) the query q is facing both ways if and only if q lies on the sub-path from $\bar{v}_i$ to $\bar{u}_i$, and

(iii) the query q is facing v if and only if q lies on the sub-path from $\bar{u}_i$ to v with $q \neq \bar{u}_i$.

Proof. Suppose $q \in P_i$ is facing u, i.e., every shortest path from q to v contains u, i.e., $d_N(q, v) = d_N(q, u) + d_N(u, v)$ and $d_N(q, v) < d_{P_i}(q, v)$. The latter implies $q \neq \bar{v}_i$, since $d_N(\bar{v}_i, v) = d_{P_i}(\bar{v}_i, v)$.



(a) A query facing u.

(b) A query facing both ways.

(c) A query facing v.

Figure 4.15: The three cases for queries in parallel-path networks. Consider the extended shortest path tree from a query $q \in P_i$ along the paths P_1 (center), P_i (bottom), and P_j (top). When q is facing u, we reach v via u. When q is facing both ways, we enter path P_1 through both terminals u and v. When q is facing v, we reach u via v. Points coloured red are reached fastest via a path through u or towards u along P_i, while points coloured blue are reached fastest through v or on a path towards v along P_i.

Moreover, $\bar{v}_i$ cannot lie on the sub-path from q to u along P_i, since otherwise

$$d_N(q,v) = d_N(q,\bar{v}_i) + d_N(\bar{v}_i,v) \overset{q\neq\bar{v}_i}{>} d_N(\bar{v}_i,v) \ ,$$

which contradicts the definition of $\bar{v}_i$ as the farthest point in N from v along the path P_i. Therefore, if q is facing u, then q lies between u and $\bar{v}_i$ along P_i with $q \neq \bar{v}_i$.

Conversely, suppose q lies between u and $\bar{v}_i$ along P_i with $q \neq \bar{v}_i$. No shortest path from q to v can contain $\bar{v}_i$ in its interior. Hence, every shortest path from q to v reaches v via u, i.e., q is facing u. Symmetrically, q is facing v if and only if $q \neq u_i$ lies between $\bar{u}_i$ and v with $q \neq \bar{u}_i$. Consequently, all points on the sub-path from $\bar{v}_i$ to $\bar{u}_i$ along P_i are facing both ways. □

Queries Facing One Way

Query points that are facing u or v only occur on u-v-paths that are strictly longer than the shortest u-v-path P_1, since $|P_j| = |P_1|$ implies $\bar{v}_j = u$ and $\bar{u}_j = v$ for all $j = 1, 2, \ldots, \lambda$.

We consider a u-facing query q from a u-v-path P_i for some $i = 2, 3, \ldots, \lambda$ with $|P_i| > |P_1|$. Every shortest path from q to any point outside of P_i leaves P_i through u. Hence, the farthest point from q on P_j with $j \neq i$ is the farthest point $\bar{u}_j$ from u on P_j and the distance from q to $\bar{q}_j$ is $d_N(q,\bar{q}_j) = d_N(q,u) + d_N(u,\bar{u}_j) = d_{P_i}(q,u) + (|P_1| + |P_j|)/2$. On the other hand, the farthest point $\bar{q}_i$ from q on the u-v-path P_i itself moves from $\bar{u}_i$ to v as q moves from u to $\bar{v}_i$ maintaining a distance of $d_N(q,\bar{q}_i) = (|P_1| + |P_i|)/2$. Therefore, the farthest distance from q in N is

$$\bar{d}_N(q) = \max\left[\frac{|P_1| + |P_i|}{2},\ d_{P_i}(q,u) + \max_{j\neq i}\left(\frac{|P_1| + |P_j|}{2}\right)\right]$$

$$= \begin{cases} d_{P_i}(q,u) + \dfrac{|P_1| + |P_\lambda|}{2} & \text{if } i \neq \lambda \\ d_{P_i}(q,u) + \dfrac{|P_1| + |P_{\lambda-1}|}{2} & \text{if } i = \lambda \text{ and } \dfrac{|P_\lambda| - |P_{\lambda-1}|}{2} \leq d_{P_i}(q,u) \\ \dfrac{|P_1| + |P_\lambda|}{2} & \text{if } i = \lambda \text{ and } \dfrac{|P_\lambda| - |P_{\lambda-1}|}{2} \geq d_{P_i}(q,u) \end{cases} .$$

To support farthest-distance queries when facing u, we compute the lengths $|P_1|, |P_2|, \ldots, |P_\lambda|$ of the u-v-paths $P_1, P_2, \ldots, P_\lambda$ as well as the distance to u for each vertex of N. This takes $O(n)$ time and allows us to evaluate $\bar{d}_N(q)$ in $O(1)$ time for every query point q that is facing u.

To support farthest-point queries when facing u, we locate the points $\bar{u}_j$ with $|P_j| = |P_\lambda|$ or $|P_j| = |P_{\lambda-1}|$ for $j = 1, 2, \ldots, \lambda$ in advance. The farthest points from u are the $\bar{u}_j$ with $|P_j| = |P_\lambda|$. We answer a farthest-point query from a point $q \in P_i$ that is facing u with $q \neq u$ as follows.

- If $i \neq \lambda$, then we report all points $\bar{u}_j$ where $|P_j| = |P_\lambda|$ for $j = 1, 2, \ldots, \lambda$ with $i \neq j$.

  The farthest points from $q \in P_i$ with $i \neq \lambda$ are the farthest points $\bar{u}_j$ from u in N along the longest u-v-paths—except for the farthest points $\bar{u}_i$ from u on P_i itself. The farthest point $\bar{q}_i$ from q in N along P_i itself is not a farthest point from q, since $q \neq u$ and, thus,

  $$\bar{d}_N(q) = d_{P_i}(q,u) + \frac{|P_1| + |P_\lambda|}{2} > \frac{|P_1| + |P_\lambda|}{2} \geq \frac{|P_1| + |P_i|}{2} = d_N(q, \bar{q}_i) .$$

- If $i = \lambda$ and $|P_\lambda| - |P_{\lambda-1}| \leq 2d_{P_i}(q,u)$, then we report all points $\bar{u}_j$ where $|P_j| = |P_{\lambda-1}|$ for $j = 1, 2, \ldots, \lambda - 1$. If $i = \lambda$ and $|P_\lambda| - |P_{\lambda-1}| \geq 2d_{P_i}(q,u)$, then we report the farthest point $\bar{q}_\lambda$ from q on P_λ itself. The overlap $|P_\lambda| - |P_{\lambda-1}| = 2d_{P_i}(q,u)$ of these cases is intentional.

  We consider a query point $q \in P_\lambda$ such that $d_{P_i}(q,u) \sim (|P_\lambda| - |P_{\lambda-1}|)/2$ for $\sim \in \{<, =, >\}$. Note that $\sim$ is $>$ when $|P_\lambda| = |P_{\lambda-1}|$, since $q \neq u$ and, thus, $d_{P_i}(q,u) > 0 = (|P_\lambda| - |P_{\lambda-1}|)/2$. Depending on $\sim$, the farthest points from q in N are ($>$) the farthest points $\bar{u}_j$ from u along the paths P_j with $|P_j| = |P_\lambda| - 1$ and $j < \lambda$, or ($<$) the farthest point $\bar{q}_\lambda$ from q along P_λ itself, or ($=$) both, because we have $d_N(q, \bar{u}_{\lambda-1}) \sim d_N(q, \bar{q}_\lambda)$, since

  $$\begin{aligned} d_N(q, \bar{u}_{\lambda-1}) &= d_{P_i}(q,u) + \frac{|P_1| + |P_{\lambda-1}|}{2} \\ &\sim \frac{|P_\lambda| - |P_{\lambda-1}|}{2} + \frac{|P_1| + |P_{\lambda-1}|}{2} \\ &= \frac{|P_1| + |P_\lambda|}{2} \\ &= d_N(q, \bar{q}_\lambda) . \end{aligned}$$

We support $O(\log n)$-time queries for $\bar{q}_\lambda$ by building the data structure for farthest point queries in the cycle $P_1 \cup P_\lambda$. This means it takes $O(k + \log n)$ time to answer farthest-point queries when facing u after $O(n)$ preprocessing, where k is the number of reported farthest points and n is the number of vertices of N. Swapping u and v yields the procedure for queries facing v.

Queries Facing Both Ways

We consider a query q that is facing both ways along a u-v-path P_i for some $i = 1, 2, \dots, \lambda$. Every path P_j with $j \neq i$ contains points that we reach from q with a shortest path via u as well as points that we reach with a shortest path from q via v. This means that there are no farthest points from q in N on P_i itself. Let $\bar{q}_j$ be the farthest point from q on the cycle $P_j \cup P_i$ for $j \neq i$. As the distance from q to $\bar{q}_j$ is $d_N(q, \bar{q}_j) = (|P_i| + |P_j|)/2$, the farthest distance from q in N is

$$\bar{d}_N(q) = \max_{j \neq i} \left(\frac{|P_i| + |P_j|}{2} \right) = \begin{cases} \dfrac{|P_i| + |P_\lambda|}{2} & \text{if } i \neq \lambda \\ \dfrac{|P_\lambda| + |P_{\lambda-1}|}{2} & \text{if } i = \lambda \end{cases} .$$

The first case applies for queries $q \in P_i$, with $i < \lambda$, where the farthest points from q lie on the longest u-v-paths, i.e., on the paths P_j with $|P_j| = |P_\lambda|$. The second case applies for queries $q \in P_\lambda$, where the farthest points from q lie on the second longest u-v-paths, i.e., on the paths P_j with $|P_j| = |P_{\lambda-1}|$ and $j < \lambda$. Using a binary search, we can answer a farthest point query from $q \in P_i$ when facing both ways by reporting the points $\bar{q}_j$ on those k paths P_j that contain farthest points from q. To improve the resulting query time of $O(k \log n)$, we take a closer look at the position of $\bar{q}_j$ relative to $\bar{u}_j$ and $\bar{v}_j$. Notice how the farthest point $\bar{q}_j$ from $q \in P_i$ along the path P_j moves from $\bar{u}_j$ to $\bar{v}_j$ as q moves from $\bar{v}_i$ to $\bar{u}_i$ along P_i, as depicted in Figure 4.16.

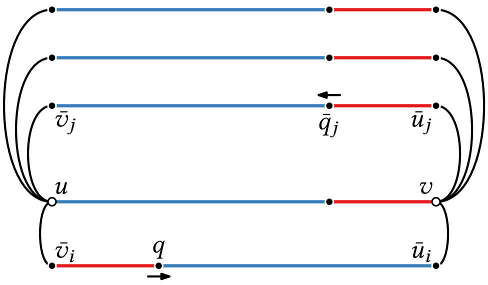


Figure 4.16: The points $\bar{v}_j$, $\bar{q}_j$, and $\bar{u}_j$ on the cycle $P_i \cup P_j$ for q between $\bar{v}_i$ and $\bar{u}_i$ with $i \neq j$.

Lemma 4.2. *Let N be a parallel-path network with terminals u and v that connect the paths $P_1, P_2, \dots, P_\lambda$ of lengths $|P_1| \leq |P_2| \leq \cdots \leq |P_\lambda|$. For each $i = 1, 2, \dots, \lambda$, the following holds.*

- *(i) The sub-path from $\bar{v}_i$ to $\bar{u}_i$ along P_i has length $d_{P_i}(\bar{v}_i, \bar{u}_i) = |P_1| = d_N(u, v)$ and the sub-paths from u to $\bar{v}_i$ and from $\bar{u}_i$ to v along P_i have length $d_{P_i}(u, \bar{v}_i) = d_{P_i}(\bar{u}_i, v) = (|P_i| - |P_1|)/2$.*
- *(ii) For every point q along the sub-path from $\bar{v}_i$ to $\bar{u}_i$, the sub-path from $\bar{v}_i$ to q has the same length as the sub-path from $\bar{u}_j$ to $\bar{q}_j$, i.e., $d_{P_i}(\bar{v}_i, q) = d_{P_j}(\bar{u}_j, \bar{q}_j)$, for any $j \neq i$.*

Proof. Since P_1 is a shortest path from u to v in N, we have $|P_1| = d_N(u, v)$ with $u = \bar{v}_1$ and $v = \bar{u}_1$. Therefore, $d_{P_1}(\bar{v}_1, \bar{u}_1) = |P_1|$ and $d_{P_1}(u, \bar{v}_1) = d_{P_1}(\bar{u}_1, v) = 0 = (|P_1| - |P_1|)/2$. For each $i = 2, \dots, \lambda$, the farthest point $\bar{u}_i$ from u in N along the path P_i is the farthest point from u along the cycle $P_1 \cup P_i$, i.e., $d_N(u, \bar{u}_i) = (|P_i| + |P_1|)/2$. Hence, the sub-path from v to $\bar{u}_i$ along P_i has length

$$d_{P_i}(v, \bar{u}_i) = d_N(u, \bar{u}_i) - d_N(u, v) = (|P_i| + |P_1|)/2 - |P_1| = (|P_i| - |P_1|)/2 \ .$$

Likewise, the sub-path from u to $\bar{v}_i$ along P_i has length $d_{P_i}(u, \bar{v}_i) = (|P_i| - |P_1|)/2$. Therefore, the sub-path from $\bar{v}_i$ to $\bar{u}_i$ along P_i has length $d_{P_i}(\bar{v}_i, \bar{u}_i) = |P_1|$, since

$$d_{P_i}(\bar{v}_i, \bar{u}_i) = |P_i| - d_{P_i}(v, \bar{u}_i) - d_{P_i}(u, \bar{v}_i) = |P_i| - (|P_i| - |P_1|) = |P_1| \enspace .$$

We show the second claim for $i = 1$. Suppose we move a point q with unit speed from $u = \bar{v}_1$ to $v = \bar{u}_1$ along P_1. For each $j = 2, 3, \ldots, \lambda$, the farthest point $\bar{q}_j$ from q along P_j moves with unit speed from $\bar{u}_j$ to $\bar{v}_j$, since q and $\bar{q}_j$ are antipodal points on the cycle $P_1 \cup P_j$. Therefore, the sub-path from $\bar{u}_1$ to q along P_1 has the same length as the sub-path from $\bar{v}_j$ to $\bar{q}_j$ along P_j.

We show the second claim for $i = 2, \ldots, \lambda$. Consider a point $q \in P_i$ on the sub-path from $\bar{v}_i$ to $\bar{u}_i$, and let $\bar{q}_j$ be the farthest point from q in N along P_j with $j \neq i$. The points q and $\bar{q}_j$ are antipodal along the cycle $P_i \cup P_j$. Let q_1 be the antipodal of $\bar{q}_j$ in the cycle $P_1 \cup P_j$. Since $u, \bar{u}_j$ and $q_1, \bar{q}_j$ are pairs of antipodal points along the cycle $P_1 \cup P_j$, we have $d_{P_1}(u, q_1) = d_{P_j}(\bar{u}_j, \bar{q}_j)$. Since $d_{P_i}(u, \bar{v}_i) = d_{P_i}(\bar{u}_i, v)$, the distance of q to $\bar{v}_i$ is the same as the distance from q_1 to $u = \bar{v}_1$, i.e., $d_{P_i}(\bar{v}_i, q) = d_{P_1}(u, q_1) = d_{P_j}(\bar{u}_j, \bar{q}_j)$. Therefore, we have $d_{P_i}(\bar{v}_i, q) = d_{P_j}(\bar{u}_j, \bar{q}_j)$ for any $i \neq j$. □

Let $J \subset \{1, 2, \ldots, \lambda\}$ be a set of indices. Suppose we search for each $\bar{q}_j$ with $j \in J$ as part of a query from $q \in P_i$ with $i \notin J$. Using Lemma 4.2, we interpret these searches as a single search *with a common key* (the distance from $\bar{v}_i$ to q), in *multiple lists* (the sub-paths from $\bar{v}_j$ to $\bar{u}_j$ for $j \in J$) of *comparable search keys* (the vertices from $\bar{v}_j$ to $\bar{u}_j$ ordered by their distance to $\bar{u}_j$).

We construct a fractional cascading data structure [18] in $O(n)$ time that supports $O(\ell + \log n)$-time predecessor queries on the sub-paths from $\bar{v}_j$ to $\bar{u}_j$ for the ℓ paths P_j where $|P_j| = |P_{\lambda-1}|$. We answer a farthest-point query from $q \in P_i$ as follows. If $i \neq \lambda$, then we locate and report $\bar{q}_\lambda$ along P_λ in $O(\log n)$ time using binary search. If $i = \lambda$ or $|P_\lambda| = |P_{\lambda-1}|$, then the remaining farthest points from q are the $\bar{q}_j$ where $j \neq i$ and $|P_j| = |P_{\lambda-1}|$; we report them in $O(k + \log n)$ time using the fractional cascading data structure. This query might report a point on P_i, which would be $\bar{q}_i$ for queries from outside P_i. For queries from within P_i, we omit this artifact.

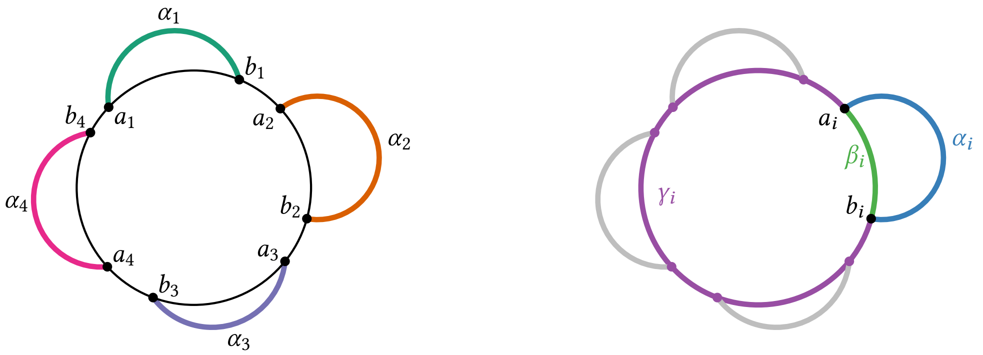
Theorem 4.3. *For every parallel-path network N with n vertices, there is a data structure with size $O(n)$ and construction time $O(n)$ that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in N, where k is the number of reported farthest points.* □

The number of farthest points in a parallel-path network N is at most the parallelism λ of N, i.e., $k \in \{1, 2, \ldots, \lambda\}$ in Theorem 4.3, as each parallel-path contains at most one farthest point.

4.3 Bead-Chain Networks

A *bead-chain network* is a two-terminal series-parallel network that is constructed as follows. First, we create a cycle by applying one parallel operation to the starting edge uv followed by a sequence of series operations. Second, we apply at most one parallel operation to each edge of this cycle to create paths that are attached to the cycle. Third, we subdivide the edges of the resulting network with a sequence of series operations. Figure 4.17a depicts a bead-chain.

This means each bead-chain network B consists of a main cycle C with paths $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$ that are non-overlapping in the following sense. For each $i = 1, 2, \ldots, \sigma$, one of the two paths along C



(a) A bead-chain network.

(b) The paths β_i and γ_i for an arc α_i.

Figure 4.17: A bead-chain network with four arcs and the three paths defined by an arc.

that connect the endpoints of α_i does not contain any endpoint of any α_j with $i \neq j$ in its interior. This means that the paths that are attached to C have a circular order. Without loss of generality, let this order be $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$. We label the endpoints of α_i with a_i and b_i such that the circular order of the endpoints along C is $a_1, b_1, a_2, b_2, \ldots, a_\sigma, b_\sigma$. As illustrated in Figure 4.17b, let β_i be the path from a_i to b_i along C that does not contain any other endpoints in its interior, and let γ_i be the path from b_i to a_i along C that contains all endpoints of the other attached paths. Without loss of generality, let α_i be at least as long as β_i. Otherwise, we swap the labels of α_i and β_i. We refer to the attached paths $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$ as the *arcs* of B.

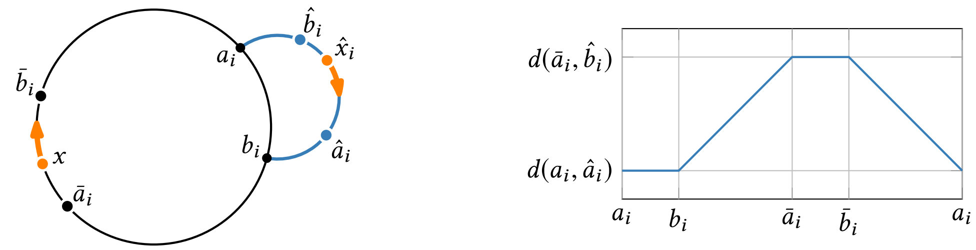
We develop a data structure for queries in bead-chain networks based on the following ideas.

1. We first develop a data structure supporting queries from the main cycle C of B; we explain later how to answer queries from any arc of B with the help of a query from C.
2. We divide a query q from C into a query for the farthest point from q on C, and into a query for the farthest points from q on the arcs of B. For the former, we rely on the data structure for cycles. For the latter, we develop a new data structure for farthest-arc queries, i.e., queries reporting the arcs that contain farthest points from q.
3. We describe the farthest distance from the points along C to each arc as a function of $x \in C$ and then construct the upper envelope of these functions. This upper envelope describes the farthest distance to any point along an arc. It is piecewise linear with line segments of slope minus one, zero, and plus one and it can be constructed in linear time.
4. We observe that the farthest arcs of a query point on C appear as a contiguous sublist of the circular list L of those arcs that are farthest-arcs for some point on C. This means we can support farthest-arc queries by constructing L and by storing a pointer to one farthest arc in L with each line segment of the upper envelope from the previous idea.

The resulting data structure supports farthest-distance queries in $O(\log n)$ time and farthest-point queries in $O(k + \log n)$-time in a bead-chain network B after $O(n)$ construction time, where n is the number of vertices of B and k is the number of reported farthest points.

4.3.1 Queries from the Main Cycle

We begin by supporting queries from the main cycle C of a bead-chain network. To support farthest-distance queries from $q \in C$, we compare the farthest distance from q in C with the farthest distance from q to any point on an arc of B and report the higher value. To support farthest-point queries from $q \in C$, we compare the farthest distance from q on C with the farthest distance from q to the arcs and then report farthest points accordingly. We use the data structure for cycles to support queries for farthest-points on C. In the following, we develop a new data structure to support queries for the farthest-points on the arcs for queries from C.



(a) The main cycle with arc α_i (blue).

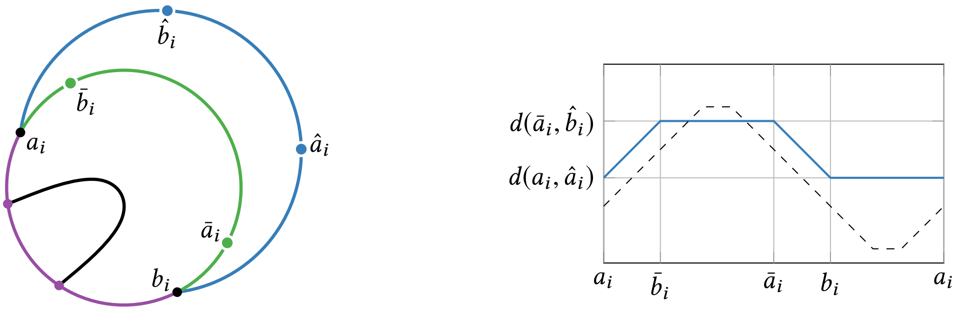
(b) A plot of the function $\hat{d}_i(x)$.

Figure 4.18: The distance $\hat{d}_i(x)$ from $x \in C$ to the farthest point $\hat{x}_i$ from x on arc α_i.

We study the farthest distance from the points along the main cycle C to any point on arc α_i, i.e., the function $\hat{d}_i(x) = \max_{y \in \alpha_i} d_B(x, y)$ for $x \in C$ and $i = 1, 2, \ldots, \sigma$. Let $\hat{x}_i$ be the farthest point from x on arc α_i in B with $\hat{a}_i := \hat{a}_{i\,i}$ and $\hat{b}_i := \hat{b}_{i\,i}$, and let $\bar{a}_i$ and $\bar{b}_i$ be the farthest points from a_i and b_i along C, respectively. To determine $\hat{d}_i(x)$, we consider the parallel-path network $C \cup \alpha_i$, since $\hat{d}_i(x) = d_B(x, \hat{x}_i) = d_{C \cup \alpha_i}(x, \hat{x}_i)$. This is because each arc α_j is at least as long as the path β_j along C for each $j = 1, 2, \ldots, \sigma$. Hence, there is a shortest path from x to $\hat{x}_i$ in B that does not contain any arc other than α_i. We derive that $\hat{d}_i(x)$ has the shape depicted in Figure 4.18. We assume, for the moment, that β_i has length at most $|C|/2$, i.e., $\bar{a}_i$ and $\bar{b}_i$ lie on γ_i.

We slide a point x along the cycle C such that we encounter a_i, b_i, $\bar{a}_i$, and $\bar{b}_i$ in this order. When x moves from a_i to b_i, the point $\hat{x}_i$ moves from $\hat{a}_i$ to $\hat{b}_i$ with a constant distance to x. From b_i to $\bar{a}_i$, the point $\hat{x}_i$ remains at $\hat{b}_i$ while the distance between $\hat{x}_i$ and x increases. From $\bar{a}_i$ to $\bar{b}_i$, the point $\hat{x}_i$ moves from $\hat{b}_i$ back to $\hat{a}_i$ with a constant distance to x. Finally, when x moves from $\bar{b}_i$ to a_i, the point $\hat{x}_i$ remains at $\hat{a}_i$ while the distance between $\hat{x}_i$ and x decreases. If x moves with unit speed, then the distance between x and $\hat{x}_i$ decreases with unit speed when x moves towards $\hat{x}_i$ and it increases with unit speed when x moves towards $\hat{x}_i$, for each $i = 1, 2, \ldots, \sigma$. This means that the plots of the functions $\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_\sigma$ consist of line segments with slope one (increasing segment), zero (low plateau, high plateau), and minus one (decreasing segment).

The height of the upper envelope $\hat{D}$ of the functions $\hat{d}_1, \ldots, \hat{d}_\sigma$ at $q \in C$ indicates the farthest distance from q to any point on the arcs $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$. In the following, we construct $\hat{D}$ in linear time using the shape of the functions $\hat{d}_1, \ldots, \hat{d}_\sigma$. One arc may need a special treatment in the construction of $\hat{D}$: we call an arc α_i *bad* when the path β_i is strictly longer than $|C|/2$. Otherwise, we call α_i *good*. Figure 4.19 illustrates a bad arc α_i with its function $\hat{d}_i$.



(a) A bad arc (blue).

(b) The farthest distance to a bad arc.

Figure 4.19: A bad arc α_i (blue) in a bead-chain network where β_i (green) is longer than the remaining cycle γ_i (purple). The shape of $\hat{d}_i$ is the same as for good arcs, but its high plateau may horizontally overlap with the high plateau of other arcs.

Lemma 4.4. *Every bead-chain network has at most one bad arc.*

Proof. Assume, for the sake of a contradiction, that there is a bead-chain network that has two bad arcs α_i and α_j with $i \neq j$. This means the length of each path β_i and β_j is greater than $|C|/2$. Therefore, β_i and β_j overlap along C, i.e., at least one endpoint of β_j lies in the interior of β_i. This contradicts our choice of β_i as the path along C that connects the endpoints of α_i and does not contain the endpoints of another arc in its interior. Thus, there is at most one bad arc. □

Lemma 4.5. *Let $\alpha_1, \dots, \alpha_\sigma$ be the good arcs of a bead-chain network B as they appear along the main cycle C of B, and let $\hat{d}_i(x)$ be the farthest distance from $x \in C$ to arc α_i for $i = 1, 2, \dots, \sigma$.*

(i) The high plateaus of $\hat{d}_1, \dots, \hat{d}_\sigma$ appear in the same order as the good arcs $\alpha_1, \dots, \alpha_\sigma$ appear along the main cycle C and no two high plateaus overlap horizontally.

(ii) The low plateaus of $\hat{d}_1, \dots, \hat{d}_\sigma$ appear in the same order as the good arcs $\alpha_1, \dots, \alpha_\sigma$ appear along the main cycle C and no two low plateaus overlap horizontally.

Proof. Let $\alpha_1, \dots, \alpha_\sigma$ be the good arcs of a bead-chain B as they appear on its main cycle C. For each $i = 1, 2, \dots, \sigma$, the farthest points $\bar{a}_i$ and $\bar{b}_i$ from the endpoints a_i and b_i of arc α_i appear along γ_i. Therefore, the points $\bar{a}_1, \bar{b}_1, \bar{a}_2, \bar{b}_2, \dots, \bar{a}_\sigma$, and $\bar{b}_\sigma$ appear in this order along C. Claim (i) follows, since the high plateau of $\hat{d}_i$ lies between $\bar{a}_i$ and $\bar{b}_i$. Claim (ii) follows, since $\hat{d}_i$ has its low plateau on β_i and since the interiors of the paths $\beta_1, \beta_2 \dots, \beta_\sigma$ are pairwise disjoint. □

We incrementally construct the upper envelope $\hat{D}$ of the functions $\hat{d}_1, \hat{d}_2, \dots \hat{d}_\sigma$ that describe the farthest distances from the main cycle C of a bead-chain B to its good arcs $\alpha_1, \dots, \alpha_\sigma$, and we treat a bad arc α_{bad} separately, if it exists. To answer a farthest-point query from $q \in C$ in B, we compare the farthest distance from q to the bad arc α_{bad} with the farthest distance $\hat{D}(q)$ from q to any good arcs. Depending on the answer, we report the farthest points from q.

Lemma 4.6. *Let $\alpha_1, \dots, \alpha_\sigma$ be the good arcs of a bead-chain network B as they appear along the main cycle C of B, and let $\hat{d}_i(x)$ be the farthest distance from $x \in C$ to arc α_i for $i = 1, 2, \dots, \sigma$. Computing the upper envelope $\hat{D}$ of the functions $\hat{d}_1, \dots, \hat{d}_\sigma$ takes $O(\sigma)$ time.*

Proof. We proceed in two passes: in the first pass, we consider only the high plateaus, the increasing segments, and the decreasing segments of $\hat{d}_1, \dots, \hat{d}_\sigma$; the respective low plateaus are replaced by extending the corresponding decreasing and increasing segments. In the second pass, we traverse the partial upper envelope from the first pass again and compare it with the previously omitted low plateaus, thereby constructing $\hat{D}$, as illustrated in Figure 4.20.

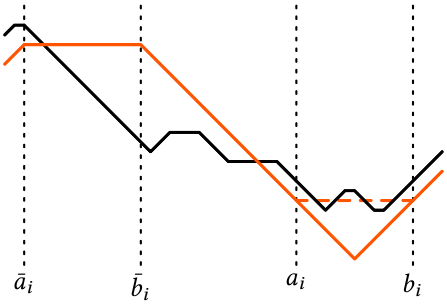


Figure 4.20: An incremental step where we construct $\hat{D}'_i$ from $\hat{D}'_{i-1}$ (black) and $\hat{d}'_i$ (orange). The treatment of the low plateau of $\hat{d}_i$ (dashed) is deferred to the second pass.

Let $\hat{d}'_i$ be the function resulting from replacing the low plateau of $\hat{d}_i$ by extending its increasing and decreasing segments and let $\hat{D}'_i$ be the upper envelope of $\hat{d}'_1, \dots, \hat{d}'_i$. In the first pass, we construct $\hat{D}'_\sigma$ incrementally, starting with $\hat{D}'_1 = \hat{d}'_1$. For $i = 2, 3, \dots, \sigma$, we obtain $\hat{D}'_i$ by inserting $\hat{d}'_i$ into $\hat{D}'_{i-1}$, as depicted in Figure 4.20. We perform this insertion by walking from $\bar{a}_i$—the left endpoint of the high plateau of $\hat{d}_i$—in both directions updating the current upper envelope $\hat{D}_{i-1}$. Locating $\bar{a}_i$ takes constant time, since $\bar{a}_i$ is the first bending point to the right of $\bar{b}_{i-1}$.

There is no more than one increasing segment of $\hat{D}'_{i-1}$ between $\bar{a}_i$ and $\bar{b}_i$. Assume, for the sake of a contradiction, that there are two increasing segments s_1 and s_2 between $\bar{a}_i$ and $\bar{b}_i$. Neither of them has their higher endpoint between $\bar{a}_i$ and $\bar{b}_i$, since there would be two horizontally overlapping high plateaus otherwise. Since s_1 and s_2 have the same slope, only one of them can be part of the upper envelope. When the segments s_1 and s_2 happen to overlap, we consider only the segment that was inserted first. Symmetrically, we treat the decreasing segments. Therefore, inserting the high plateau of $\hat{d}_i$ into the upper envelope $\hat{D}'_{i-1}$ takes constant time.

If the decreasing segment of $\hat{d}'_i$ appears along $\hat{D}'_i$ at all, then it appears at its highest point at $\bar{b}_i$. We update the previous upper envelope $\hat{D}'_{i-1}$ by walking from $\bar{b}_i$ towards a_i until the decreasing segment of $\hat{d}'_i$ vanishes below $\hat{D}'_{i-1}$. We charge the costs for this walk to the segments that are removed from the previous upper envelope $\hat{D}'_{i-1}$. Symmetrically, we process the increasing segment of $\hat{d}'_i$ by walking from $\bar{a}_i$ towards b_i. Each increasing segment and each decreasing segment appears at most once along any intermediate upper envelope and is never considered again after its removal. Therefore, the total cost for inserting all increasing and decreasing segments—and, thus, the total cost of constructing $\hat{D}'_\sigma$—amounts to $O(\sigma)$.

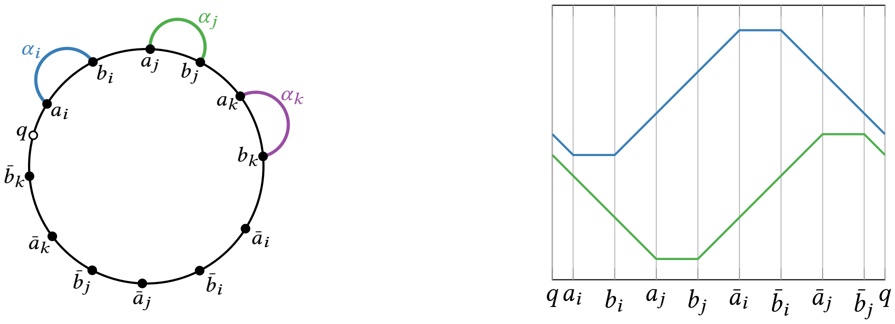
In the second pass, we construct the upper envelope $\hat{D}$ from $\hat{D}'_\sigma$. Since no two low plateaus overlap, we walk along $\hat{D}'_\sigma$ comparing its height to the height of the current low plateau, if any. This takes $O(\sigma)$ time, since there are σ low plateaus and $\hat{D}'_\sigma$ consists of $O(\sigma)$ line segments. □

Consider a bead-chain network B where all arcs are good. To answer a farthest-point query from a point q on the main cycle C of B, we need to find the farthest arcs from q, i.e., the arcs that contain farthest points from q in B. If $\hat{D}$ has a plateau at q, then q has at most two farthest arcs: one arc α_i where $\hat{d}_i$ has a low plateau at q and one arc α_j where $\hat{d}_j$ has a high plateau at q. This means that we can store these at most two farthest arcs directly with each plateau of $\hat{D}$. If the farthest-arc distance $\hat{D}$ has an increasing or decreasing segment at q, then several arcs could contain farthest points from q in B. We rely on the following observations to locate these arcs.

An arc α of B is considered *relevant* when there exists some point $x \in C$ such that α is a farthest arc for x and α is considered *irrelevant* when there is no such point on the cycle C.

Lemma 4.7. *Let α_i, α_j, and α_k be arcs that appear in this order along the main cycle C of a bead-chain network B without a bad arc. If α_i and α_k are farthest arcs from a point $q \in C$ such that $\hat{d}_i$ and $\hat{d}_k$ are both decreasing at q, then α_j is either also a farthest arc from q or α_j is irrelevant.*

Proof. Suppose α_j is not a farthest arc from q in B. We show $\hat{d}_j(x) < \hat{d}_i(x)$ for all $x \in C$, which means that α_j is irrelevant. As illustrated in Figure 4.21, q lies between a_i and $\bar{b}_k$, since $\hat{d}_i$ and $\hat{d}_k$ are both decreasing at q. Hence, q lies between a_j and $\bar{b}_j$, i.e., $\hat{d}_j$ is decreasing at q, as well.



(a) The three arcs.

(b) The comparison of $\hat{d}_i$ (blue) and $\hat{d}_j$ (green).

Figure 4.21: The constellation from Lemma 4.7, where α_i and α_k are farthest arcs from q where the farthest distance decreases as q moves in clockwise direction (a). A comparison of the arc distance functions $\hat{d}_i$ and $\hat{d}_j$ reveals that α_j is irrelevant in this case (b).

Let $\Delta(x) := \hat{d}_i(x) - \hat{d}_j(x)$. We have $\Delta(a_i) = \Delta(q) > 0$, as $\hat{d}_i$ and $\hat{d}_j$ are decreasing with the same slope from q to a_i. We observe that $\Delta(b_i) = \Delta(a_i) + d(a_i, b_i)$, as $\hat{d}_i$ remains constant from a_i to b_i while $\hat{d}_j$ decreases. We have $\Delta(a_j) = \Delta(b_i) + 2d(b_i, a_j)$, as $\hat{d}_i$ increases from b_i to a_j while $\hat{d}_j$ decreases. We continue in this fashion and obtain the following description of Δ.

$$\begin{aligned}
\Delta(a_i) &= \Delta(q) & \Delta(\bar{a}_i) &= \Delta(b_j)\\
\Delta(b_i) &= \Delta(a_i) + d(a_i, b_i) & \Delta(\bar{b}_i) &= \Delta(\bar{a}_i) - d(\bar{a}_i, \bar{b}_i)\\
\Delta(a_j) &= \Delta(b_i) + 2d(b_i, a_j) & \Delta(\bar{a}_j) &= \Delta(\bar{b}_i) - 2d(\bar{b}_i, \bar{a}_j)\\
\Delta(b_j) &= \Delta(a_j) + d(a_j, b_j) & \Delta(\bar{b}_j) &= \Delta(\bar{a}_j) - d(\bar{a}_j, \bar{b}_j)
\end{aligned}$$

This implies $\Delta(x) \geq \Delta(q) > 0$ for all $x \in C$, since $d(a_i, b_i) = d(\bar{a}_i, \bar{b}_i)$, $d(b_i, a_j) = d(\bar{b}_i, \bar{a}_j)$, and $d(a_j, b_j) = d(\bar{a}_j, \bar{b}_j)$. Therefore, the arc α_j is irrelevant, since for every point x along the cycle C, the farthest point $\hat{x}_i$ from x on α_i is further away than the farthest point $\hat{x}_j$ from x on α_j. □

Corollary 4.8. *Let q be a point on the main cycle C of a bead-chain network B without a bad arc. The farthest arcs α from q where the farthest distance from C to any point on the arc α is locally increasing or decreasing near q form a contiguous sub-list in the circular list of relevant arcs.* □

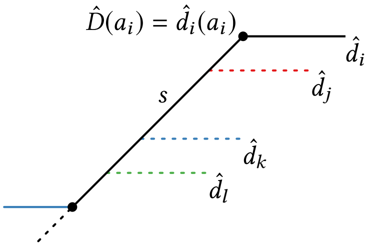


Figure 4.22: An increasing segment s along $\hat{D}$ where the functions $\hat{d}_j$, $\hat{d}_k$, $\hat{d}_l$ fall below $\hat{D}$ and only $\hat{d}_i$ remains at the upper endpoint.

When we traverse an increasing segment s along $\hat{D}$, then the set of functions among $\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_\sigma$ that define $\hat{D}$ shrinks until only one function $\hat{d}_i$ remains at the upper endpoint of s, as illustrated in Figure 4.22. Otherwise, two high plateaus of $\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_\sigma$ would overlap at a_i. The arc α_i is a farthest arc for all points $x \in C$ such that $\hat{D}(x) \in s$, and we can find the remaining farthest arcs from x by traversing the circular list of relevant arcs of B starting from α_i, due to Corollary 4.8.

Using this observation, we can support farthest-arc queries from the main cycle of a bead-chain network B without a bad arc as follows. First, we determine the farthest-arc distance function $\hat{D}$ alongside with the circular list L of the relevant arcs in B. Each increasing segment s along the plot of $\hat{D}$ stores a pointer to the arc α in L that defines $\hat{D}$ at the higher endpoint of the segment s. Likewise, we store a pointer to the arc in L that defines the upper endpoint of each decreasing line segment along the plot of $\hat{D}$. As depicted in Figure 4.23, we subdivide each horizontal segment s of $\hat{D}$ depending on which low plateaus or high plateaus overlap along s and store the at most two arcs that determine the subdivided segments directly with them. This construction takes $O(n)$ time, since we obtain all relevant information during the construction of $\hat{D}$, and it supports $O(l + \log n)$-time farthest-arc queries from the main cycle C of the bead-chain B, where l is the number of reported farthest arcs.

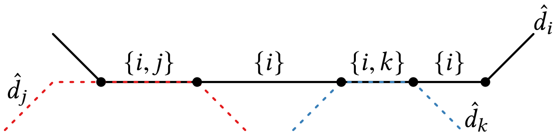


Figure 4.23: A horizontal segment s along $\hat{D}$, where the low plateau of $\hat{d}_i$ overlaps with the high plateaus of $\hat{d}_j$ and $\hat{d}_k$. We subdivide the segment s depending on the functions $\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_\sigma$ that determine $\hat{D}$ and store their indices with the subdivided segments.

Lemma 4.9 (Main Cycle Queries). *Let B be a bead-chain network with n vertices, σ good arcs, and main cycle C. There exists a data structure with $O(n)$ construction time that supports $O(\log n)$-time farthest-distance queries in B from the main cycle C and $O(k + \log n)$-time farthest-point queries in B from C, where $k \in \{1, 2, \ldots, \sigma + 2\}$ is the number of reported farthest points.*

Proof. Let B be a bead-chain with main cycle C, good arcs $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$, and a bad arc α_{bad}.

The data structure for queries from the main cycles consists of two parts: The first part is the data structure for queries in the parallel-path network $C \cup \alpha_{\text{bad}}$ to determine the farthest distance and farthest points from $q \in C$ to points on the main cycle C itself and on the bad arc α_{bad}. The second part is the data structure for farthest-arc queries from C in the bead-chain network $B \setminus \alpha_{\text{bad}}$ to determine farthest points from q on the good arcs $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$. The overall construction time is $O(n)$, since it takes takes $O(n)$ time to decompose B into the main cycle, the good arcs, and the bad arc, and since constructing each part of the data structure takes $O(n)$ time.

We answer a farthest-distance query from $q \in C$ by reporting the larger of the farthest distance from q in $C \cup \alpha_{\text{bad}}$ and the farthest-arc distance from q in $B \setminus \alpha_{\text{bad}}$. This takes $O(\log \sigma)$ time, as the queries in $C \cup \alpha_{\text{bad}}$ and $B \setminus \alpha_{\text{bad}}$ take $O(1)$ and $O(\log \sigma)$ time, respectively.

To answer a farthest-point query from $q \in C$, we first perform a farthest-distance query from q to determine whether we need to report farthest points from $C \cup \alpha_{\text{bad}}$, or $B \setminus \alpha_{\text{bad}}$, or both. We report the at most two farthest points from q in $C \cup \alpha_{\text{bad}}$ with a farthest-point query from q in $C \cup \alpha_{\text{bad}}$ that takes constant time. We report the good arcs that contain farthest points from q using a farthest-arc query from q in $B \setminus \alpha_{\text{bad}}$. This query takes $O(k + \log \sigma)$ time, where k is the number of arcs with farthest points from q. We could locate each farthest point from q on an arc with a binary search. This would lead to a query time of $O(k \log n + \log \sigma) = O(k \log n)$.

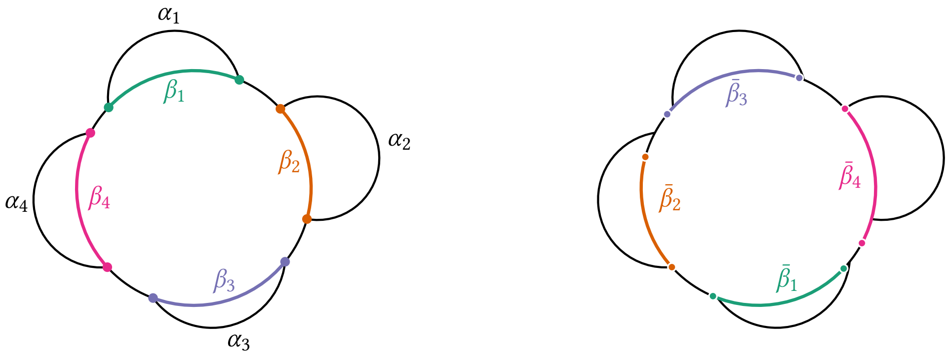
To reduce the query time to $O(k + \log n)$, we review some observations about the farthest point from $q \in C$ on a good arc α_i. Recall that β_i is the path along C that connects a_i and b_i and that does not contain the endpoints of any other arc in its interior, and recall that γ_i is the other path along C that connects a_i and b_i. Let $\bar{a}_i$ and $\bar{b}_i$ be the farthest points from a_i and b_i, respectively, along the main cycle C and let $\hat{a}_i$ and $\hat{b}_i$ be the farthest points from a_i and b_i, respectively, along arc α_i. Since α_i is a good arc, β_i is at most as long as γ_i and, thus, $\bar{a}_i, \bar{b}_i \in \gamma_i$. Therefore, we encounter the points $a_i, b_i, \bar{a}_i, \bar{b}_i$ in this circular order along C. Recall the following.

1. If q lies between a_i and b_i, then the farthest point $\hat{q}_i$ from q on α_i lies on the path from $\hat{b}_i$ to $\hat{a}_i$. More precisely, $\hat{q}_i$ is the farthest point from q along the cycle $\alpha_i \cup \beta_i$.
2. If q lies between b_i and $\bar{a}_i$, then $\hat{b}_i$ is the farthest point from q on α_i.
3. If q lies between $\bar{a}_i$ and $\bar{b}_i$, then the farthest point $\hat{q}_i$ from q on α_i lies on the path from $\hat{b}_i$ to $\hat{a}_i$. More precisely, $\hat{q}_i$ is the farthest point from q along the cycle $\alpha_i \cup \gamma_i$.
4. If q lies between $\bar{b}_i$ and a_i, then $\hat{a}_i$ is the farthest point from q on α_i.

The interiors of the paths $\beta_1, \beta_2, \ldots, \beta_\sigma$ are pairwise disjoint, by definition. Let $\bar{\beta}_i$ be the shorter path from $\bar{b}_i$ to $\bar{a}_i$ along C. Since $\bar{\beta}_i$ is a mirrored copy of β_i in C, the interiors of the paths $\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_\sigma$ are pairwise disjoint, as depicted in Figure 4.24. Hence, every query point $q \in C$ lies in at most one path β_i and in at most one path $\bar{\beta}_j$ for some $i, j = 1, 2, \ldots, \sigma$ with $i \neq j$. This means that there are at most two good arcs where we need to locate the farthest point from q on these arcs; on all other arcs, locating the farthest point from q takes constant time. Therefore, reporting the farthest points from q on the good arcs takes $O(k + \log n)$ time. □

4.3.2 Queries from the Good Arcs

Consider a bead-chain network B with main cycle C, good arcs $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$ and a bad arc α_{bad}. For a farthest-distance query q from a good arc α_i, we determine the farthest distance from q to any point on the cycle $\alpha_i \cup \beta_i$ and the farthest distance from q in B to any point on the remainder



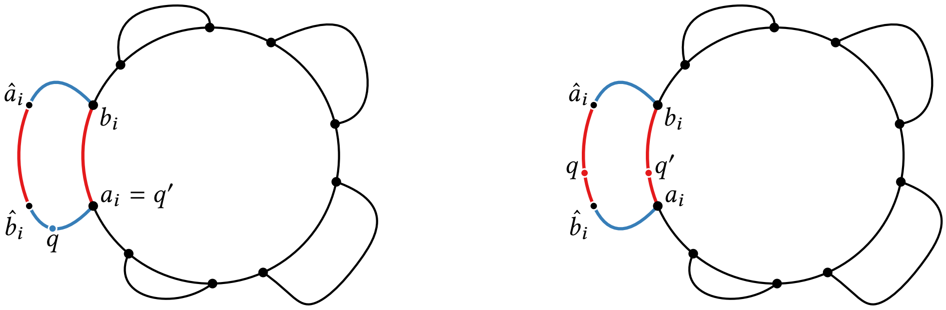
(a) The paths $\beta_1, \beta_2, \ldots, \beta_\sigma$.

(b) The paths $\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_\sigma$.

Figure 4.24: The disjoint paths $\beta_1, \beta_2, \ldots, \beta_\sigma$ and their mirrored copies $\bar{\beta}_1, \bar{\beta}_2, \ldots, \bar{\beta}_\sigma$.

$B \setminus (\alpha_i \cup \beta_i)$ of the network. For a farthest-point query from $q \in \alpha_i$, we report the farthest points from q in $\alpha_i \cup \beta_i$ or $B \setminus (\alpha_i \cup \beta_i)$, depending on which are farther away.

The data structure for queries from the good arcs consists of (1) the data structures for queries in the cycles $\alpha_1 \cup \beta_1, \alpha_2 \cup \beta_2, \ldots, \alpha_\sigma \cup \beta_\sigma$ and (2) the data structure for queries from C in B. We explain how we use the latter data structure to find the farthest points from a query q on a good arc α_i in $B \setminus (\alpha_i \cup \beta_i)$. The overall construction takes $O(n)$ time for a bead-chain with n vertices, since the cycles $\alpha_1 \cup \beta_1, \alpha_2 \cup \beta_2, \ldots, \alpha_\sigma \cup \beta_\sigma$ are pairwise edge-disjoint, and since constructing the data structure for queries from C in B takes $O(n)$ time, as shown in Lemma 4.9.



(a) A query facing a_i.

(b) A query facing both ways.

Figure 4.25: Projecting a query q from an arc α_i onto a point q' along the path β_i. When q is facing a_i, as in Subfigure (a), then q' is a_i. When q is facing both ways, as in Subfigure (b), q' is the point along β_i that splits β_i at the same ratio as q splits α_i.

We project a query from $q \in \alpha_i$ with $a_i \neq q \neq b_i$ for the farthest points in $B \setminus (\alpha_i \cup \beta_i)$ to a query from some point q' along β_i, as illustrated in Figure 4.25. This translation depends on the position of q along α_i. Recall that $\hat{a}_i$ and $\hat{b}_i$ are the farthest points from a_i and b_i along α_i with respect to B. The points $\hat{a}_i$ and $\hat{b}_i$ are the antipodals of a_i and b_i along the cycle $\alpha_i \cup \beta_i$. We distinguish the cases from parallel-path networks: The query point q is *facing* a_i when every shortest path in B from q to b_i contains a_i, the query point q is *facing* b_i when every shortest path in B from q to a_i contains b_i, and the query point q is *facing both ways* otherwise.

1. The query q is facing a_i if and only if q lies on the sub-path from a_i to $\hat{b}_i$ with $q \neq \hat{b}_i$.

   If q is facing a_i, then the farthest distance from q to any points on $B \setminus \alpha_i$ is the distance from q to a_i plus the farthest distance from a_i in the bead-chain network $B \setminus \alpha_i$, i.e.,

   $$\bar{d}_B(q) = \max\left(\bar{d}_{\alpha_i \cup \beta_i}(q),\ d_B(q, a_i) + \bar{d}_{B \setminus \alpha_i}(a_i)\right) \quad .$$

   We distinguish two cases depending on whether a_i has any farthest points in $B \setminus \alpha_i$.

   a) Suppose a_i has some farthest point in B that does not lie on α_i.

      This means we have $\bar{d}_{B \setminus \alpha_i}(a_i) = \bar{d}_B(a_i)$ and, thus,

      $$\bar{d}_B(q) = \max\left(\bar{d}_{\alpha_i \cup \beta_i}(q),\ d_B(q, a_i) + \bar{d}_B(a_i)\right) \quad .$$

      We answer a farthest-distance query from q with three constant-time queries: a query for the farthest distance $\bar{d}_{\alpha_i \cup \beta_i}(q)$ from q in $\alpha_i \cup \beta_i$, a query for the distance $d_B(q, a_i) = d_{\alpha_i}(q, a_i)$ between q and a_i along α_i, and a query for the farthest distance $\bar{d}_B(a_i)$ from a_i in B. If a farthest-distance query from q indicates that q has farthest points in $B \setminus \alpha_i$, then we report these farthest points in $O(k + \log n)$ time with a farthest-point query from a_i in B. If this query reports $\hat{a}_i$, then we omit this point from the output. If a farthest-distance query from q indicates that q has a farthest point in B on $\alpha_i \cup \beta_i$, then we find it in $O(\log n)$ time with a query from q in $\alpha_i \cup \beta_i$.

   b) Suppose all farthest points from a_i in B lie on α_i, i.e., only $\hat{a}_i$ is farthest from a_i in B.

      This means we have $\bar{d}_{B \setminus \alpha_i}(a_i) < \bar{d}_B(a_i)$. In this case, we cannot use a query from a_i to obtain the information that we need to answer a query from $q \in \alpha_i$.

      We resolve this issue by constructing the data structure for queries from C in the bead-chain network $B \setminus \alpha_i$. We show in Lemma 4.10 that B has at most one arc α_i where some point on β_i has only farthest points on α_i. This means that this case occurs at most once and the overall construction time remains linear.

      We answer queries from q in the same manner as in the previous case, with the only difference that we perform all queries from a_i in $B \setminus \alpha_i$, instead of in B.

2. The query q is facing b_i if and only if q lies on the sub-path from $\hat{a}_i$ to b_i with $q \neq \hat{a}_i$.

   We handle queries facing b_i symmetrically to queries facing a_i.

3. The query q is facing both ways if and only if q lies on the sub-path from $\hat{b}_i$ to $\hat{a}_i$.

   We translate the query from q to a query from β_i. By Lemma 4.2, the sub-path from $\hat{b}_i$ to $\hat{a}_i$ along α_i has the same length as β_i and both the sub-path from a_i to $\hat{b}_i$, and the sub-path from $\hat{a}_i$ to b_i have length $(|\alpha_i| - |\beta_i|)/2$. Let q' be the point on β_i such that $d(a_i, q') = d(\hat{b}_i, q)$, i.e., the relative position of q' to a_i and b_i along β_i is the same as the relative position of q to $\bar{b}_i$ and $\bar{a}_i$ along α_i. We call q' the *projection* of q from α_i onto β_i.

For any point $g \in B \setminus (\alpha_i \cup \beta_i)$, we have $d(q,g) = d(q',g) + (|\alpha_i| - |\beta_i|)/2$, because the lengths of the shortest paths from q to g and from q' to g differ by the length of the path from $\bar{a}_i$ to b_i. This means that the farthest distance from q in B is

$$\bar{d}_B(q) = \max\left(\bar{d}_{\alpha_i \cup \beta_i}(q),\ \bar{d}_{B \setminus \alpha_i}(q') + \frac{|\alpha_i| - |\beta_i|}{2}\right) .$$

More precisely, we have $\bar{d}_B(q) = \bar{d}_{B\setminus\alpha_i}(q') + (|\alpha_i| - |\beta_i|)/2$, since

$$\begin{aligned} 2\bar{d}_{B\setminus\alpha_i}(q') + |\alpha_i| - |\beta_i| &\geq 2\bar{d}_C(q') + |\alpha_i| - |\beta_i| \\ &= |\alpha_i| + |C| - |\beta_i| \\ &= |\alpha_i| + |\gamma_i| \\ &\geq |\alpha_i| + |\beta_i| \\ &= 2\bar{d}_{\alpha_i \cup \beta_i}(q) \ , \end{aligned}$$

which means that if $\alpha_i \cup \beta_i$ contains a farthest point from q in B, then $|\beta_i| = |\gamma_i| = |C|/2$.

We distinguish two cases depending on whether q' has any farthest points in $B \setminus \alpha_i$.

a) Suppose q' has some farthest point in B that does not lie on α_i.

   This means we have $\bar{d}_{B\setminus\alpha_i}(a_i) = \bar{d}_B(a_i)$ and, thus,

   $$\bar{d}_B(q) = \bar{d}_{B\setminus\alpha_i}(q') + (|\alpha_i| - |\beta_i|)/2 = \bar{d}_B(q') + (|\alpha_i| - |\beta_i|)/2 \ .$$

   We answer a farthest-distance query from q by adding the precomputed values $(\alpha_i - \beta_i)/2$ to the answer of a farthest-distance query from q' in B that takes $O(\log n)$ time. We answer a farthest-point query from q in $O(k + \log n)$ time with a farthest-point query from q' in B. If this query reports a farthest point on α_i, then we omit this point from the output. If $|\beta_i| = |\gamma_i|$, then we check if we need to report an additional farthest point from q along the cycle $\alpha_i \cup \beta_i$ with a query from q in $\alpha_i \cup \beta_i$.

b) Suppose all farthest points from q' in B lie on α_i, i.e., only $\hat{q}_i$ is farthest from q' in B.

   This means we have $\bar{d}_{B\setminus\alpha_i}(q') < \bar{d}_B(q')$. In this case, we cannot use a query from q' to obtain the information that we need to answer a query from q.

   As before, we resolve this issue by constructing the data structure for queries from C in the bead-chain network $B \setminus \alpha_i$. We argue in Lemma 4.10 below that this case applies for at most one arc and, thus, the construction time remains linear.

   We answer queries from q in the same manner as in the previous case, with the only difference that we perform all queries from q' in $B \setminus \alpha_i$ instead of in B.

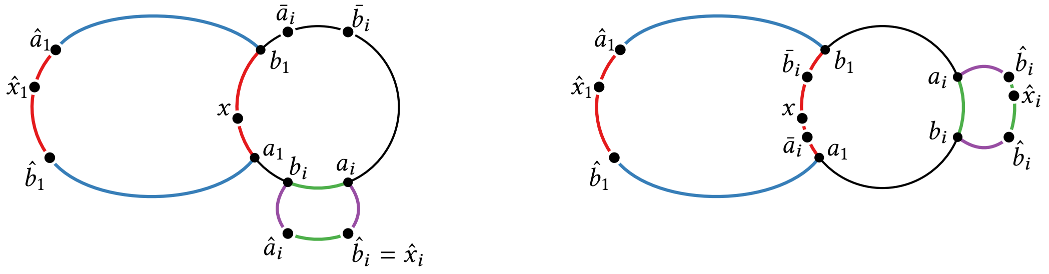
An arc α_i is *long* when there is a point $x \in \beta_i$ such that α_i is the only farthest arc of x in B. We have described a data structure supporting $O(\log n)$-time farthest-distance and $O(k + \log n)$-time farthest-point queries from the good arcs of a bead-chain network. The construction time of this data structure is $O(n + ln)$, where l is the number of good long arcs of B. We show $l \leq 1$.

Lemma 4.10. *Every bead-chain network as at most one long good arc.*

Proof. Let B be a bead-chain network with main cycle C and good arcs $\alpha_1, \alpha_2, \dots, \alpha_\sigma$. Suppose α_1 is a long arc. We argue that the cycle $\alpha_1 \cup \beta_1$ is strictly longer than any other cycle $\alpha_i \cup \beta_i$ for $i = 2, \dots, \sigma$. This implies that B cannot have more than one long arc that is good.

Since α_1 is a long arc, there exists a point $x \in \beta_1$ such that α_1 is the only farthest arc from x in B. This means that the farthest point $\hat{x}_1$ from x on α_1 with respect to B is strictly farther away from x than the farthest point $\hat{x}_i$ from x on any other arc α_i for any $i = 2, 3, \dots, \sigma$.

Consider a good arc α_i for some $i = 2, 3, \dots, \sigma$. The point x lies on γ_i, since $x \in \beta_1$ and since the interiors of the paths $\beta_1, \beta_2, \dots, \beta_\sigma$ are pairwise disjoint. Let $\bar{a}_i$ and $\bar{b}_i$ be the farthest points on C from the endpoints a_i and b_i of the arc α_i, respectively. Since α_i is a good arc, we know that β_i is at most as long as γ_i and, thus, $\bar{a}_i, \bar{b}_i \in \gamma_i$. More precisely, $b_i, \bar{a}_i, \bar{b}_i, a_i$ appear in this order along γ_i. We distinguish three cases based on the position of x along γ_i.



(a) The point x lies between $\bar{a}_i$ and b_i.

(b) The point x lies between $\bar{b}_i$ and $\bar{a}_i$.

Figure 4.26: Two positions of the witness x for α_1 being long along γ_i for some $i \geq 2$.

1. Suppose x lies on the sub-path from b_i to $\bar{a}_i$ along γ_i with $x \neq \bar{a}_i$, as in Figure 4.26a.

   This means that x is facing b_i in the parallel-path network $C \cup \alpha_i$, i.e., every shortest path in B from x to the farthest point $\hat{x}_i$ from x on α_i passes through b_i. This implies that the farthest point $\hat{x}_i$ from x on α_i is $\hat{b}_i$, i.e., the farthest point from b_i on α_i. Since α_i is a good arc, β_i is at most as long as γ_i and, thus, $\hat{b}_i$ is the farthest point from b_i on the cycle $\alpha_i \cup \beta_i$. Therefore, the cycle $\alpha_1 \cup \beta_1$ is strictly longer than the cycle $\alpha_i \cup \beta_i$, because

$$\begin{aligned}
|\alpha_1 \cup \beta_1|/2 &= d_B(x, \hat{x}_1) && (x \text{ and } \hat{x}_1 \text{ are antipodal on } \alpha_1 \cup \beta_1)\\
&> d_B(x, \hat{x}_i) && (x \text{ is a witness for } \alpha_1 \text{ being long})\\
&\geq d_B(b_i, \hat{x}_i) && (\text{the shortest path from } x \text{ to } \hat{x}_i \text{ contains } b_i)\\
&= d_B(b_i, \hat{b}_i) && (\hat{x}_i = \hat{b}_i, \text{ since } x \text{ is facing } b_i \text{ in } C \cup \alpha_i)\\
&= |\alpha_i \cup \beta_i|/2 \ . && (b_i, \hat{b}_i \text{ are antipodal on } \alpha_i \cup \beta_i)
\end{aligned}$$

2. Suppose x lies on the sub-path from $\bar{a}_i$ to $\bar{b}_i$ along γ_i, as depicted in Figure 4.26b.

   This means that x is facing both ways in the parallel-path network $C \cup \alpha_i$, i.e., the farthest point $\hat{x}_i$ from x on α_i in B is the farthest point from x along the cycle $\gamma_i \cup \alpha_i$. Therefore, the cycle $\alpha_1 \cup \beta_1$ is strictly longer than the cycle $\alpha_i \cup \beta_i$, because

$$\begin{aligned}
|\alpha_1 \cup \beta_1|/2 &= d_B(x, \hat{x}_1) && (x \text{ and } \hat{x}_1 \text{ are antipodal on } \alpha_1 \cup \beta_1)\\
&> d_B(x, \hat{x}_i) && (x \text{ is a witness for } \alpha_1 \text{ being long})\\
&= |\gamma_i \cup \alpha_i|/2 && (x \text{ and } \hat{x}_i \text{ are antipodal on } \gamma_i \cup \alpha_i)\\
&\geq |\beta_i \cup \alpha_i|/2 \ . && (\alpha_i \text{ is a good arc})
\end{aligned}$$

3. Suppose x lies on the sub-path from $\bar{b}_i$ to a_i along γ_i with $x \neq \bar{b}_i$.

   This means that x is facing a_i in the parallel-path network $C \cup \alpha_i$, i.e., every shortest path in B from x to the farthest point $\hat{x}_i$ from x on α_i passes through a_i. The claim $|\alpha_1 \cup \beta_1| > |\alpha_i \cup \beta_i|$ follows symmetrically to the case where x is facing b_i in $C \cup \alpha_i$.

The above implies $|\alpha_1 \cup \beta_1| > |\alpha_i \cup \beta_i|$ for any $i = 2, \ldots, \sigma$. If there was any other good long arc α_i with $i \neq 1$ in B, then we would have the contradiction $|\alpha_1 \cup \beta_1| > |\alpha_i \cup \beta_i| > |\alpha_1 \cup \beta_1|$. Therefore, every bead-chain network has at most one good arc that is long. ☐

Since each bead-chain network has at most one good long arc, we obtain the following.

Lemma 4.11 (Queries from the Good Arcs). *Let B be a bead-chain network with n vertices and σ good arcs. There is a data structure with $O(n)$ construction time that supports farthest-distance queries in B from the good arcs of B in $O(\log n)$ time and farthest-point queries in B from the good arcs in $O(k + \log n)$ time, where $k \in \{1, 2, \ldots, \sigma + 2\}$ is the number of reported farthest points.* ☐

4.3.3 Queries from the Bad Arc

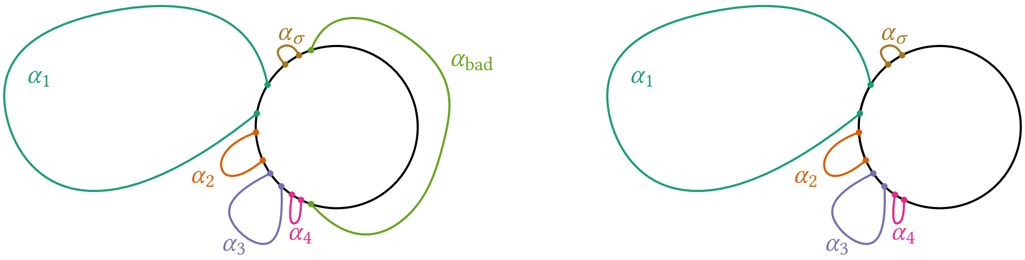
We complete the data structure for bead-chain networks by discussing queries from a bad arc. Figure 4.27 shows the sub-networks that we build to support queries in a bead-chain.

Theorem 4.12. *Let B be a bead-chain network with n vertices and σ good arcs. There is a data structure with $O(n)$ size and $O(n)$ construction time supporting $O(k + \log n)$-time farthest-point queries on B, where $k \in \{1, 2, \ldots, \sigma + 2\}$ is the number of reported farthest points.*

Proof. Consider a bead-chain network B with main cycle C, with one bad arc α_{bad}, and with σ good arcs $\alpha_1, \ldots, \alpha_\sigma$. Let $B \setminus \alpha_{\text{bad}}$ be the network obtained by removing the bad arc from B. We support queries from the main cycle and from the good arcs using the data structures from Lemmas 4.9 and 4.11. We support queries from the bad arc α_{bad} as follows.

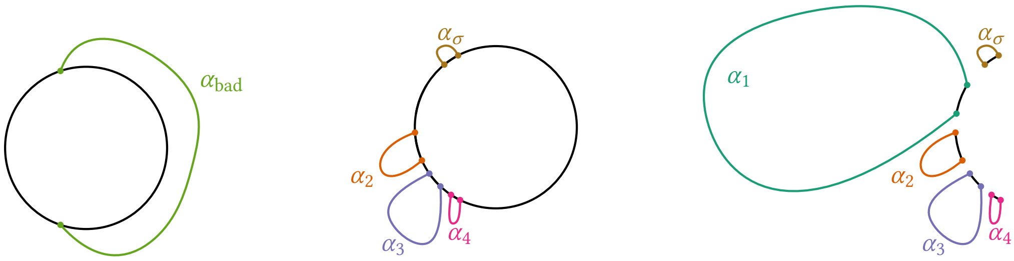
Recall that β_{bad} is the path along C that connects the endpoints of α_{bad} and that does not contain the endpoints of any other arc in its interior, and recall that γ_{bad} is the other path along C connecting the endpoints of α_{bad}. Since α_{bad} is a bad arc, we have $|\gamma_{\text{bad}}| < |\beta_{\text{bad}}|$. Any shortest path from $q \in \alpha_{\text{bad}}$ to a farthest point $\bar{q}$ from q in B traverses β_{bad} only if $\bar{q}$ lies on β_{bad} itself.

The farthest points from a query $q \in \alpha_{\text{bad}}$ are the farthest points from q in the parallel-path network $\alpha_{\text{bad}} \cup C = \alpha_{\text{bad}} \cup \beta_{\text{bad}} \cup \gamma_{\text{bad}}$, or the farthest points from q in the bead-chain $B \setminus \beta_{\text{bad}}$, or both. Notice that $B \setminus \beta_{\text{bad}}$ is a bead-chain network without a bad arc and α_{bad} is part of the main



(a) The bead-chain B.

(b) The network $B \setminus \alpha_{\text{bad}}$.



(c) $C \cup \alpha_{\text{bad}}$

(d) $C \cup \alpha_2 \cup \cdots \cup \alpha_\sigma$

(e) The cycles $\alpha_1 \cup \beta_1, \alpha_2 \cup \beta_2, \ldots, \alpha_s \cup \beta_s$.

Figure 4.27: The sub-networks that we create to support queries in a bead-chain B with main cycle C, good arcs $\alpha_1, \alpha_2, \ldots, \alpha_\sigma$ and a bad arc α_{bad}, where α_1 is a good long arc.

cycle of $B \setminus \beta_{\text{bad}}$. We support queries from the bad arc α_{bad} by constructing the data structure for $\alpha_{\text{bad}} \cup C$ from Theorem 4.3 and the data structure for $B \setminus \beta_{\text{bad}}$ from Lemma 4.9.

The entire data structure for queries in B has $O(n)$ construction time, since each edge of B appears only in a constant number of data structures with linear construction time. Every farthest-point query takes $O(k + \log n)$ time, because each query consists of a constant number of $O(\log n)$-time farthest-distance queries followed by a constant number of farthest point queries in those sub-structures that actually contain farthest points from the original query. For each query, we report at most one farthest point on each arc and at most one farthest-point on the main cycle. Therefore, the number of reported farthest points ranges from 1 to $\sigma + 2$. □

When we use the data structure for queries in a bead-chain as part of another data structure, then we are sometimes in a position to preempt the binary searches during the queries: We subdivide the bead-chain network in advance. We introduce a new vertex (1) at the farthest point along the main cycle C from every vertex of B, (2) at the farthest point on the cycle $\alpha_i \cup \beta_i$ from every vertex along the arc α_i, and (3) at every point along the main cycle C where a linear piece of the farthest distance function $\bar{d}_B(x) = \max_{y \in B} d_B(x, y)$, for $x \in C$, ends.

Corollary 4.13. *Let B be a bead-chain with n vertices. We can subdivide B with $O(n)$ vertices in $O(n)$ time such that the resulting bead-chain B' satisfies the following. There is a data structure with $O(n)$ size and $O(n)$ construction time supporting $O(1)$-time farthest-distance queries on B' and $O(k)$-time farthest-point queries on B', where k is the number of reported farthest points.* □

4.4 Abacus Networks

An *abacus network* is a two-terminal series-parallel network that is generated as follows. First, we generate a parallel path network X. Second, we apply at most one parallel operation for each edge of X. Third, we subdivide the resulting network using a sequence of series operations. As illustrated in Figure 4.28, an abacus A consists of a parallel-path network N with terminals u and v whose parallel paths $P_1, P_2, \dots, P_\lambda$ have paths attached to them that satisfy the following:

1. The endpoints a and b of each attached path α are vertices on the same u-v-path of N.
2. The attached paths are non-overlapping along each u-v-path P_i of N in the sense that if a and b are the endpoints of a path α that is attached to P_i, then the path β that connects a and b in P_i does not contain the endpoints of any other attached path in its interior.
3. Without loss of generality, each path α that is attached to P_i is at least as long as the path β in P_i that connects the endpoints of α along P_i in N. Otherwise, we swap α and β.

We refer to the paths that are—in the above sense—attached to a parallel-path network N in an abacus network A as the *arcs* of A. For $i = 1, 2, \dots, \lambda$, let B_i be the path P_i with its arcs in A. We refer to $B_1, B_2, \dots, B_\lambda$ as the *bead-strings* of A. Note that any pair of bead-strings B_i and B_j of an abacus A, with $i \neq j$, form a bead-chain network $B_i \cup B_j$. In the following, we assume that the paths $P_1, P_2, \dots, P_\lambda$ of N are sorted by their weighted lengths $|P_1| \leq |P_2| \leq \dots \leq |P_\lambda|$.

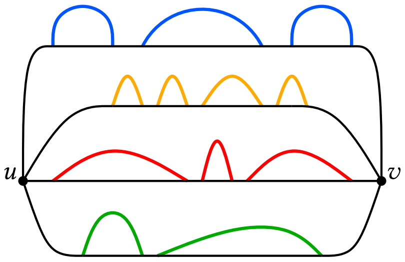
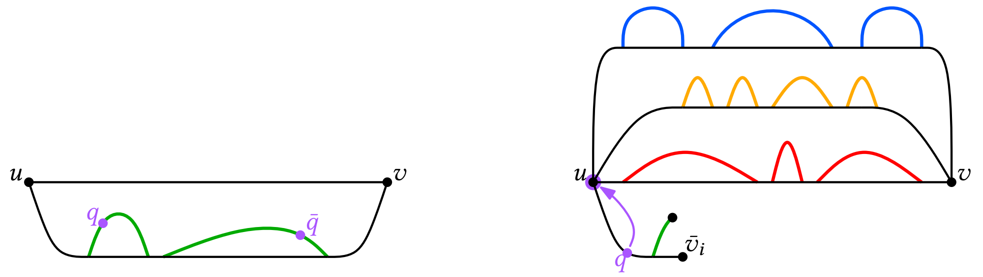


Figure 4.28: An abacus with arcs (coloured) attached to its parallel-path network.

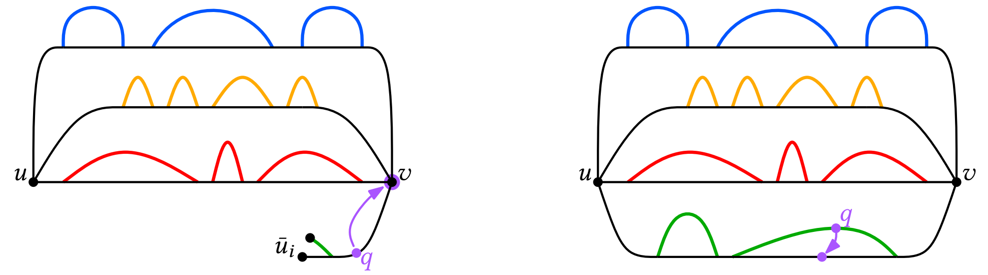
If an abacus network A has only few parallel paths, then we support queries on A as follows. For each $i, j = 1, 2, \dots, \lambda$ with $i \neq j$, we support queries in the bead chain $B_i \cup B_j$. This construction takes $O(\lambda n)$ time, since each edge of A appears in λ bead-chains. To answer a farthest-point query from $q \in B_i$, we either report the farthest points from u or v (for queries facing one terminal) or we perform a farthest-distance query from q in each $B_i \cup B_j$ (for queries facing both ways). This takes $O(\lambda \log n)$ time and reveals which bead-chains contain farthest points from q. Then, we report the farthest points from q in A in $O(k + \lambda \log n)$ time with at most λ farthest-point queries in the appropriate bead-chains. This data structure is optimal when $\lambda = O(1)$, i.e., when the abacus A has few parallel paths. Next, we consider the case when λ is not a constant.

We split farthest-point queries in an abacus into an *inward query* and an *outward query*: an inward query considers farthest points on the bead-string containing the query point; an outward query considers farthest points on the remaining bead-strings. We first perform the farthest distance version of inward and outward queries before reporting farthest points where appropriate. Figure 4.29 illustrates how we treat inward and outward queries.



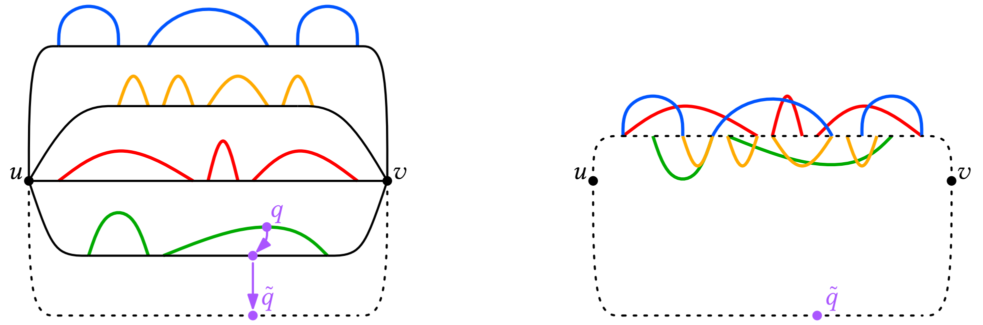
(a) An inward query.

(b) The left case of an outward query.



(c) The right case of an outward query.

(d) An outward query from an arc.



(e) Translating a query to the virtual edge.

(f) Collapsing the chains.

Figure 4.29: Inward (a) and outward (b–f) queries for the abacus network from Figure 4.28. Inward queries are answered in the bead-chain containing the query (a). When facing a terminal, outward queries are answered with queries form the terminal that the query is facing (b,c). When facing both ways, outward queries from arcs are translated to queries from the path (d) and outward queries from the path are translated to queries from a virtual edge (e). We conceptually collapse all bead-chains of the abacus to support queries from the virtual edge (f).

4.4.1 Inward Queries

An inward query from a point q along a bead-string B_i in an abacus A reports the farthest points from q in A on B_i itself, i.e., we report the points $\bar{q} \in B_i$ with $d_A(q, \bar{q}) = \max_{q' \in B_i} d_A(q, q')$. For $i = 2, 3, \ldots, \lambda$, let B'_i be the bead-chain network that consists of the bead-string B_i and an edge of length $|P_1| = d_A(u, v)$ that connects the terminals u and v. Let B'_1 be the bead-chain that consists of B_1 and an edge of length $|P_2|$. As depicted in Figure 4.29a, the network B'_i preserves distances for points on B_i with respect to A, i.e., $d_A(x, y) = d_{B'_i}(x, y)$ for all $x, y \in B_i$ and $i = 1, 2, \ldots, \lambda$. The farthest points from q along B_i in A are the farthest points from q along B_i in B'_i, i.e.,

$$\forall q \in B_i : \max_{x \in B_i} d_A(q, x) = \max_{x \in B_i} d_{B'_i}(q, x) \quad .$$

We support inward queries in an abacus A by constructing the bead-chains $B'_1, B'_2, \ldots, B'_\lambda$ as well as the data structures for farthest-distance queries and farthest-point queries therein. This construction takes $O(n)$ time and allows us to answer inward farthest-point queries on the abacus A in $O(k + \log n)$ time and inward farthest-distance queries in $O(\log n)$ time, where n is the number of vertices of A and k is the number of reported inward farthest points.

4.4.2 Outward Queries

An outward query from a point q along a bead-string B_i in an abacus A reports the farthest points from q in A on all bead-strings other than B_i, i.e., we report the points $\bar{q} \in A \setminus B_i$ with $d_A(q, \bar{q}) = \max_{q' \in A \setminus B_i} d_A(q, q')$. We distinguish the same cases as for parallel-path networks: The query q is facing u when every shortest path tree from q reaches u before v, the query q is facing v when every shortest path tree from q reaches v before u, and the query q is facing both ways otherwise. Analogously to Lemma 4.1, the query is facing u precisely when q is within distance $d(u, \bar{v}_i)$ from u, and it is facing v precisely when q is within distance $d(v, \bar{u}_i)$ from v.

An outward query from $q \in B_i$ that is facing u has the same farthest points as u outside of B_i. During the construction of the networks $B'_1, \ldots, B'_\lambda$ for inward queries, we determine a list L_j of the farthest points from u in B'_j. Similarly to the approach for parallel-path networks, we keep the lists that achieve the largest farthest distance from u. If there is only one such list L_l, then we also keep the lists that achieve the second highest farthest distance from u for queries from B_l. With this preparation, answering the query for $q \in B_i$ amounts to reporting the entries of the appropriate lists L_j with $j \neq i$. Symmetrically, we proceed when q is facing v. This means that we can perform outward farthest-point queries in $O(k)$ time and outward farthest-distance queries in $O(1)$ time, after $O(n)$ preprocessing time, when the query is facing one way.

Supporting outward queries that are facing both ways is the most challenging component of the data structure for abacus networks. It is the bottleneck of the construction time of $O(n \log \lambda)$—all other types of queries can be handled in linear time. We iterate through a number of ideas and inefficient intermediate data structures before we arrive at a data structure with size $O(n)$ and construction time $O(n \log \lambda)$ that supports $O(\log n)$-time outward farthest-distance queries and $O(k + \log n)$ outward farthest-point queries for queries that are facing both ways.

Projecting Queries from the Arcs to Queries on the Parallel-Path Network

We translate outward queries that are facing both ways from the arcs of an abacus A to outward queries from the corresponding parallel path in N, as illustrated in Figure 4.29d.

Let α be an arc with endpoints a and b along a bead-string B_i, for some $i = 1, 2, \dots, \lambda$, and let β be the path along P_i connecting a and b. Recall that α is at least as long as β. Let $\hat{a}$ and $\hat{b}$ be the farthest points along the cycle $\alpha \cup \beta$ from a and from b, respectively. The path from $\hat{b}$ to $\hat{a}$ along α has length $|\beta|$ while the path from a to $\hat{b}$ and the path from $\hat{a}$ to b both have length $(|\alpha| - |\beta|)/2$. We adapt the strategy for answering queries from an arc of a bead-chain:

- If q lies on the path from a to $\hat{b}$ with $q \neq \hat{b}$, then we reach any farthest point from q outside of B_i through a. Hence, we translate the query from q to a query from $q' = a$. Furthermore, the farthest distance from q to any point in A outside of B_i is
$$\max_{x \in A \setminus B_i} d_A(q, x) = d_A(q, a) + \max_{x \in A \setminus B_i} d_A(a, x) \ .$$
- If q lies on the path from $\hat{b}$ to $\hat{a}$, then we reach any farthest point from q outside of B_i with a shortest path through a and with a shortest path through b. Thus, the outward farthest points remain the same when we project q from α onto the point $q' \in \beta$, where q' is the relative position of q' to a and b is the relative position of q to $\hat{b}$ and $\hat{a}$, i.e., $d(\hat{b}, q) = d(a, q')$. Furthermore, the farthest distance from q to any point in A outside of B_i is
$$\max_{x \in A \setminus B_i} d_A(q, x) = d_A(\hat{b}, a) + \max_{x \in A \setminus B_i} d_A(q', x) = (|\alpha| - |\beta|)/2 + \max_{x \in A \setminus B_i} d_A(q', x) \ .$$
- If q lies on the path from $\hat{a}$ to b with $q \neq \hat{a}$, then we reach any farthest point from q outside of B_i through b. Hence, we translate the query from q to a query from $q' = b$.

This means it suffices to consider queries from the underlying parallel-path network N.

Suppose we construct data structures for queries in each bead-chain network $P_i \cup B_j$ for $i \neq j$. According to the above, we could use these data structures to report all outward farthest points from a query $q \in B_i$ in $O(k + \lambda \log n)$ time with a query in each $P_i \cup B_j$ with $i \neq j$. The overall construction time for this approach would be $O(\lambda n)$. Note that the relative position of the query point to the terminals matters, whereas the edge containing the query point does not.

Projecting Queries to a Virtual Edge

We remove the dependence on the parallel path from where the query originates. As illustrated in Figure 4.29e, we introduce a *virtual edge* $\tilde{e}$ from u to v of length $|P_1| = d_A(u, v)$, i.e., the length of the shortest u-v-path in N. By Lemma 4.2, the sub-path from $\bar{u}_i$ to $\bar{v}_i$ along P_i has the same length as $\tilde{e}$, for each $i = 1, 2, \dots, \lambda$. For a query point $q \in P_i$ let $\tilde{q}$ be the unique point on $\tilde{e}$ such that $\tilde{q}$ has the same distance to u and to v as q to $\bar{u}_i$ and to $\bar{v}_i$, i.e., $d_A(\bar{v}_i, q) = d_{\tilde{e} \cup A}(u, \tilde{q})$ and $d_A(q, \bar{u}_i) = d_{\tilde{e} \cup A}(\tilde{q}, v)$. We refer to $\tilde{q}$ as the *projection* of q onto the virtual edge $\tilde{e}$.

Lemma 4.14. *Let A be an abacus network A with parallel paths $P_1, P_2, \ldots, P_\lambda$ and bead-strings $B_1, B_2, \ldots, B_\lambda$. Consider a query point $q \in P_i$ that is facing both ways and let $\tilde{q}$ be the projection of q onto the virtual edge $\tilde{e}$ of A. The farthest points from q in $P_i \cup B_j$ are the farthest points from $\tilde{q}$ in $\tilde{e} \cup B_j$ for any $j = 1, 2, \ldots, \lambda$ with $j \neq i$ and $\bar{d}_{P_i \cup B_j}(q) = \bar{d}_{\tilde{e} \cup B_j}(\tilde{q}) + (|P_i| - |P_1|)/2$.*

Proof. Recall that P_i is at least as long as $\tilde{e}$. Suppose we continuously shrink P_i until P_i is as long as $\tilde{e}$, while maintaining the relative position of q to u and v. The distance from q to all points on B_j in $P_i \cup B_j$ increases uniformly for any $j \neq i$. Therefore, the farthest points from q in $P_i \cup B_j$ remain the same throughout the deformation and, thus, match those of $\tilde{q}$ in $\tilde{e} \cup B_j$.

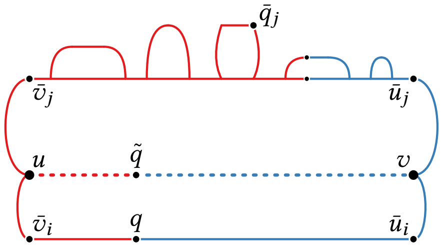


Figure 4.30: An illustration of how farthest points on bead-string B_j (top) are preserved when projecting a query q from P_i (bottom) to $\tilde{q}$ along the virtual edge $\tilde{e}$ (middle).

Let $\bar{q}_j \in B_j$ be a farthest point from q in $P_i \cup B_j$. Since q is facing both ways, there is a shortest path from q to $\bar{q}_j$ in $P_i \cup B_j$ that passes through $\bar{v}_i$ and u. Without loss of generality, assume that a shortest path from $\tilde{q}$ to $\bar{q}_j$ in $\tilde{e} \cup B_j$ passes through u, as depicted in Figure 4.30. Otherwise, we swap u and v. We have $d_{P_i}(\bar{v}_i, u) = (|P_1| + |P_i|)/2$ according to Lemma 4.2, and therefore

$$\begin{aligned}
\bar{d}_{P_i \cup B_j}(q) &= d_{P_i \cup B_j}(q, \bar{q}_j) && (\bar{q}_j \text{ is farthest from } q \text{ in } P_i \cup B_j)\\
&= d_{P_i}(q, \bar{v}_i) + d_{P_i \cup B_j}(\bar{v}_i, \bar{q}_j) && (\bar{v}_i \text{ lies on a shortest path from } q \text{ to } \bar{q}_j)\\
&= d_{P_i}(q, \bar{v}_i) + d_{P_i}(\bar{v}_i, u) + d_{B_j}(u, \bar{q}_j) && (u \text{ lies on a shortest path from } \bar{v}_i \text{ to } \bar{q}_j)\\
&= d_{\tilde{e}}(\tilde{q}, u) + d_{P_i}(\bar{v}_i, u) + d_{B_j}(u, \bar{q}_j) && (\tilde{q} \text{ is the projection of } q \text{ onto } \tilde{e})\\
&= d_{\tilde{e} \cup B_j}(\tilde{q}, \bar{q}_j) + d_{P_i}(\bar{v}_i, u) && (u \text{ lies on a shortest path from } \tilde{q} \text{ to } \bar{q}_j \text{ in } \tilde{e} \cup A)\\
&= \bar{d}_{\tilde{e} \cup B_j}(\tilde{q}) + d_{P_i}(\bar{v}_i, u) && (\bar{q}_j \text{ is farthest from } \tilde{q} \text{ in } \tilde{e} \cup B_j)\\
&= \bar{d}_{\tilde{e} \cup B_j}(\tilde{q}) + \frac{|P_i| - |P_1|}{2}\ . && (\text{by Lemma 4.2})
\end{aligned}$$

This implies that $\bar{d}_{P_i \cup B_j}(q) = \bar{d}_{\tilde{e} \cup B_j}(\tilde{q}) + (|P_i| - |P_1|)/2$. □

Since Lemma 4.14 does not depend on the index j, the outward farthest distance from $q \in P_i$ in the abacus A is equal to the farthest distance from $\tilde{q}$ in the abacus $\tilde{e} \cup (A \setminus B_i)$, plus the correction term $(|P_i| - |P_1|)/2$ that accounts for the length differences between P_i and $\tilde{e}$, i.e.,

$$\max_{x \in A \setminus B_i} d_A(q, x) = \max_{j=1, j \neq i}^{\lambda} \max_{x \in B_j} d_{P_i \cup B_j}(q, x) = \max_{j=1, j \neq i}^{\lambda} \bar{d}_{P_i \cup B_j}(q) = \frac{|P_i| - |P_1|}{2} + \max_{j=1, j \neq i}^{\lambda} \bar{d}_{\tilde{e} \cup B_j}(\tilde{q})\ .$$

We construct the data structure for queries in the bead-chains $\tilde{e} \cup B_1, \tilde{e} \cup B_2, \ldots, \tilde{e} \cup B_\lambda$. This construction takes $O(n)$ time, since each edge of A appears in at most one of these networks. We answer an outward farthest-distance query from $q \in P_i$ by projecting q onto the point $\tilde{q}$ along the virtual edge $\tilde{e}$ and performing a farthest-distance query from $\tilde{q}$ in each bead-chain $\tilde{e} \cup B_j$ with $j = 1, 2, \ldots, \lambda$ and $j \neq i$. We then report the largest value reported from the queries from $\tilde{q}$ plus the correction term $(|P_i| - |P_1|)/2$ as the farthest distance from q to any point in $A \setminus B_i$. This takes $O(\lambda \log n)$ time. We report any outward farthest points from q with farthest-point queries from $\tilde{q}$ in those bead-chains $\tilde{e} \cup B_j$ where the maximum distance from $\tilde{q}$ was attained. This means it takes $O(k + \lambda \log n)$ time to report all k outward farthest-points from q.

Superimposing the Farthest Distances to other Bead-Strings

We wish to avoid inspecting every bead-chain $\tilde{e} \cup B_j$ with $j \neq i$ for an outward farthest-point query from $q \in P_i$. For any $i = 1, 2, \ldots, \lambda$ and $\tilde{q} \in \tilde{e}$, let $\tilde{d}_i(\tilde{q}) := \bar{d}_{\tilde{e} \cup B_i}(\tilde{q})$. The farthest distance from $q \in P_i$ to any point on $A \setminus B_i$ is the upper envelope $\tilde{D}_i$ of the functions $\tilde{d}_j$ with $j \neq i$ at the projection $\tilde{q}$ of q onto $\tilde{e}$ plus the term $(|P_i| - |P_1|)/2$ that corrects the projection, i.e.,

$$\max_{x \in A \setminus B_i} d_A(q, x) = \frac{|P_i| - |P_1|}{2} + \max_{j=1, j \neq i}^{\lambda} \bar{d}_{\tilde{e} \cup B_j}(\tilde{q}) = \frac{|P_i| - |P_1|}{2} + \max_{j=1, j \neq i}^{\lambda} \tilde{d}_j(\tilde{q}) = \frac{|P_i| - |P_1|}{2} + \tilde{D}_i(\tilde{q}) \ .$$

If we construct the upper envelopes $\tilde{D}_i$ for each $i = 1, 2, \ldots, \lambda$, then we can answer queries for the outward farthest distance from $q \in P_i$ in $O(\log n)$ time by projecting q onto $\tilde{e}$ and evaluating $(|P_i| - |P_1|)/2 + \tilde{D}_i(\tilde{q})$. This construction requires $O(\lambda n \log \lambda)$ time and $O(\lambda n)$ space, if we compute each upper envelope $\tilde{D}_i$ in $O(n \log \lambda)$ time using plane sweep.

Lemma 4.15. *For any $i = 1, 2, \ldots, \lambda$, it takes $O(n \log \lambda)$ time to construct the upper envelope $\tilde{D}_i$ of the functions $\tilde{d}_j$ with $j \neq i$, and the plot of the upper envelope $\tilde{D}_i$ consists of $O(n)$ line segments.*

Proof. Let $i \in \{1, 2, \ldots, \lambda\}$. Each of the functions $\tilde{d}_1, \tilde{d}_2, \ldots, \tilde{d}_\lambda$ is piecewise linear, where each linear piece has slope minus one, zero, or one, and the total number of pieces is $O(n)$.

For each slope $x \in \{-1, 0, +1\}$, we determine the partial upper envelope $\tilde{D}_i^x$ of all line segments of slope x of the functions $\tilde{d}_j$ with $j \neq i$. We sweep a vertical line over the plane maintaining a heap of the line segments of slope x in vertical order along the sweep line. Since line segments of the same slope never change their vertical order, we only need to process the $O(n)$ endpoints of the line segments as events. Since the size of the heap never exceeds λ, processing each event takes $O(\log \lambda)$ time. Therefore, the sweep for $\tilde{D}_i^x$ takes $O(n \log \lambda)$ time.

Once we have the partial upper envelopes D_i^{-1}, D_i^0, and D_i^{+1}, we determine $\tilde{D}_i$ in a final sweep. The events of this sweep are endpoints of line segments and intersections of $\tilde{D}_i^{-1}$, $\tilde{D}_i^0$, and $\tilde{D}_i^{+1}$. Every intersection of $\tilde{D}_i^{-1}$, $\tilde{D}_i^0$, and $\tilde{D}_i^{+1}$ consumes one line segment that cannot be part of any other intersection. For instance, suppose a line segment of slope zero is currently the highest line segment and suppose it intersects a line segment of slope plus one. Then, the line segment of slope plus one cannot intersect any other line segment of D_i^0 or D_i^{-1}, afterwards. This means the final sweep has $O(n)$ events that can be processed in constant time each. Therefore, computing the upper envelope $\tilde{D}_i$ of $\tilde{d}_j$ with $j \neq i$ takes $O(n \log \lambda)$ time and consumes $O(n)$ space. □

Reporting Farthest Points on other Bead-Strings using Interval Stabbing

Aside from the upper envelopes $\tilde{D}_1, \tilde{D}_2, \ldots, \tilde{D}_\lambda$ we develop a second data structure to support queries to identify the bead-strings that contain outward farthest points from a query point. There is an outward farthest point from $q \in P_i$ on B_j with $i \neq j$ when $\tilde{d}_j(\tilde{q}) = \tilde{D}_i(\tilde{q})$, since

$$\frac{|P_i| - |P_1|}{2} + \tilde{d}_j(\tilde{q}) = \bar{d}_{P_i \cup B_j}(q) = \max_{x \in A \setminus B_i} d_A(q, x) = \frac{|P_i| - |P_1|}{2} + \tilde{D}_i(\tilde{q}) \ .$$

For each bead-string B_j with $j \neq i$, let $I_i(j)$ be the set of points $\tilde{q} \in \tilde{e}$ where $\tilde{q}$ is the projection of a point $q \in P_i$ such that B_j contains outward farthest points from q, i.e., $I_i(j) = \{\tilde{q} \in \tilde{e} \mid \tilde{d}_j(\tilde{q}) = \tilde{D}_i(\tilde{q})\}$. If the bead-string B_j has σ_j arcs, then $I_i(j)$ consists of $O(\sigma_j)$ sub-edges of $\tilde{e}$. We interpret these sub-edges as intervals on the real line within $[0, 1]$ by parameterizing $\tilde{e}$ with a function $\phi : [0, 1] \to \tilde{e}$ where $\phi(\tau)$ is the unique point on $\tilde{e}$ with distance $\tau \cdot |P_1|$ from u for any $\tau \in [0, 1]$.

We obtain the sets $I_i(j)$ for all $j = 1, 2, \ldots, \lambda$ with $j \neq i$ in linear time when constructing the upper envelope $\tilde{D}_i$ of the functions $\tilde{d}_j$ for all $j \neq i$. We say the intervals in the set $I_i(j)$ have color j. Let $I_i = \bigcup_{j=1, j \neq i}^{\lambda} I_i(j)$ collect all intervals. We construct a data structure that supports interval stabbing queries on I_i, i.e., that reports the intervals in I_i that contain a query $\tilde{q} \in \tilde{e}$. Each stabbed interval indicates a bead-string that contains outward farthest points from q. Each bead-string is reported at most once, since intervals of the same color are pairwise disjoint.

Theorem 4.16 (Coloured Interval Stabbing). *Let I be a set of n intervals in $[0, 1]$ where each interval in I has a color $c \in \{1, 2, \ldots, \lambda\}$ and all intervals of the same color have pairwise disjoint interiors. Suppose for each color c, we are given the endpoints of the intervals of color c in ascending order. There is a data structure with size $O(n)$ and construction time $O(n \log \lambda)$ that supports interval stabbing queries in I in $O(k + \log n)$ time, where k is the number of reported intervals.*

Proof. We sort the $O(n)$ endpoints of all intervals in $O(n \log \lambda)$ time using λ-way merge. Then, we apply a filtering search technique called *window lists* [17] to support stabbing queries in I. A window list for I consists of a set of intervals $[x_1, x_2], [x_2, x_3], \ldots, [x_l, x_{l+1}]$ with $0 = x_1 < x_2 < \cdots < x_l < x_{l+1} = 1$ together with *windows* $W_1, W_2, \ldots, W_l$ where each window W_i is a list that stores the intervals in I that overlap $[x_i, x_{i+1}]$, for $i = 1, 2, \ldots, l$. Moreover, each window satisfies the density condition stating that the largest number of intervals in I stabbed by some value $x \in [x_i, x_{i+1}]$ is at most twice the lowest number of intervals in I stabbed by some $x' \in [x_i, x_{i+1}]$. We construct a window list for I by processing the endpoints of the intervals in I in ascending order and adding new intervals to the current list W_i while keeping track of the number of stabbed intervals. Whenever the density condition becomes violated, we begin a new window. Chazelle [17] shows that this construction takes $O(n)$ time and consumes $O(n)$ space.

To answer an interval stabbing query $x \in [0, 1]$ with the window lists for I, we first determine the index i such that $x_i \leq x \leq x_{i+1}$ using a binary search. We know that all intervals in I that contain x are stored in W_i. Therefore, we report all intervals from W_i that contain x and we filter the remaining intervals from the output. The list W_i contains at most $2k$ intervals, where k is the number of intervals that contain x. Hence, the total query time is $O(k + \log n)$. □

The interval stabbing data structure for I_i allows us to answer outward farthest-point queries from $q \in P_i$ in $O(k + l \log n)$ time, where l is the number of bead-strings containing outward farthest points: First, we determine the colors of the intervals in I_i that are stabbed by $\tilde{q}$. When $\tilde{q}$ stabs an interval of color j, we report the outward farthest points from q in B_j with a farthest query from $\tilde{q}$ in $\tilde{e} \cup B_j$. The query in $\tilde{e} \cup B_j$ takes $O(k_j + \log n)$ time, where k_j is the number of reported farthest points in B_j. Therefore, the total time for the query is $O(k + l \log n)$.

Preempting Binary Searches from Queries in the Bead-Chains

We reduce the query time to $O(k + \log n)$. Recall that answering a farthest point query from $\tilde{q}$ in $\tilde{e} \cup B_j$ involves three sub-queries that take $O(\log n)$ time: The first sub-query reveals whether $\tilde{q}$ has a farthest point on the main cycle, on the arcs, or both. The second sub-query locates a farthest point on the main cycle, if it exists. The third and forth sub-queries locate the at most two farthest points on arcs that need to be located; the other farthest points on arcs can be computed in advance. We break each interval in $I_i(j)$ into smaller intervals depending on the answers to these sub-queries. This is done by locating the farthest point from each vertex of $\tilde{e} \cup B_j$ along its main cycle and subdividing the corresponding edge. This does not increase the asymptotic construction time, since each vertex of B causes at most one break.

If we build an interval stabbing data structure for the subdivided intervals, then it takes $O(k_j)$ time to report the farthest points from $\tilde{q}$ in $\tilde{e} \cup B_j$, since the answers to all $O(\log n)$-time sub-queries are stored with the subdivided intervals. Therefore, the overall time to report all outward farthest points from q reduces to $O(k + \log n)$. The overall construction time is $O(\lambda n \log n)$: First, we spend $O(n)$ time to build the data structure for queries in the bead-chains $\tilde{e} \cup B_1, \tilde{e} \cup B_2, \ldots, \tilde{e} \cup B_\lambda$. Then, we spend $O(\lambda n \log n)$ time to build the upper envelopes $D_i(\tilde{q}) = \max_{j=1, j \neq i}^{\lambda} \bar{d}_{\tilde{e} \cup B_j}(\tilde{q})$, for each $i = 1, 2, \ldots, \lambda$. Finally, we spend $O(\lambda n \log n)$ time to construct the interval stabbing data structures for the subdivided intervals I_i that indicate the farthest colors from points on P_i. Next, we reduce the construction time to $O(n \log n)$.

Superimposing all Bead-String Data Structures with a Double Bottom Approach

Let F be a set of real valued functions on a common domain X. The *first layer* $\mathcal{L}^1$ of the functions in F is their upper envelope, i.e., $\mathcal{L}^1(x) := \sup_{f \in F} f(x)$ for all $x \in X$. The *second layer* $\mathcal{L}^2$ of F is the partial function of the second highest functions in F, i.e., for all $x \in X$ we have

$$\mathcal{L}^2(x) := \sup_{f \in F : f(x) < \mathcal{L}^1(x)} f(x) \ .$$

The final data structure for outward farthest-distance and outward farthest-point queries for queries that are facing both ways is constructed as follows.

1. We introduce a virtual edge $\tilde{e}$ of length $|P_1|$ to A, i.e., the length of $\tilde{e}$ is the length of the shortest of the parallel paths that connect the terminals u and v of A.
2. For each $i = 1, 2, \ldots, \lambda$, we calculate the correction term $(|P_i| - |P_1|)/2$ that is required when translating farthest-distance queries from the path P_i to the virtual edge $\tilde{e}$.
3. For each $i = 1, 2, \ldots, \lambda$, we create a copy X_i of each bead-chain network $\tilde{e} \cup B_i$. For each vertex x of the copy X_i, we introduce a new vertex at the farthest point $\bar{x}$ from x along the main cycle of X_i. Note that we do not subdivide the original virtual edge $\tilde{e}$.

4. For each $i = 1, 2, \dots, \lambda$, we construct the data structure for queries in X_i. This takes $O(n)$ total time and supports $O(1)$-time farthest-distance queries and $O(k)$-time farthest-point queries in X_i, provided that queries are specified in terms of the edges of X_i.
5. For each $i = 1, 2, \dots, \lambda$, we determine the function $\tilde{d}_i$ that maps points on $\tilde{e}$ to their farthest distance in $\tilde{e} \cup B_i$, i.e., $\tilde{d}_i(\tilde{q}) = \max_{x \in \tilde{e} \cup B_i} d_{\tilde{e} \cup B_i}(x, \tilde{q})$ for any $\tilde{q} \in \tilde{e}$. We obtain these functions during the construction of the data structure for queries in $X_1, X_2, \dots, X_\lambda$.
6. We construct the first layer $\mathcal{L}^1$ and the second layer $\mathcal{L}^2$ of all functions $\tilde{d}_1, \tilde{d}_2, \dots, \tilde{d}_\lambda$. This takes $O(n \log \lambda)$ time using plane sweep. Alongside with the layers, we determine the sets $I^1(i) = \{\tilde{q} \in \tilde{e} \mid \tilde{d}_i(\tilde{q}) = \mathcal{L}^1(\tilde{q})\}$ and $I^2(i) = \{\tilde{q} \in \tilde{e} \mid \tilde{d}_i(\tilde{q}) = \mathcal{L}^2(\tilde{q})\}$ for each $i = 1, 2, \dots, \lambda$.
7. For every $i = 1, 2, \dots, \lambda$, we break each interval in $I^1(i)$ and in $I^2(i)$ into smaller intervals, depending on the subdivision of the edges in X_i. This ensures that the points in each subdivided interval have their farthest points on the same edges of X_i and, thus, on the same edges of $\tilde{e} \cup B_j$. Each subdivided interval stores its color i and a pointer to the corresponding edge in X_i. Let I^1 be the set of the subdivided intervals from $I^1(1), I^1(2), \dots, I^1(\lambda)$, and let I^2 be the set of subdivided intervals from $I^2(1), I^2(2), \dots, I^2(\lambda)$. This process takes $O(n)$ time, since each vertex of A breaks at most one interval.
8. We construct an interval stabbing data structure for the intervals in I^1, and we construct an interval stabbing data structure for the intervals in I^2. This takes $O(n \log \lambda)$ time.

The entire construction requires $O(n \log \lambda)$ time and consumes $O(n)$ space. The reason why we need the second layer becomes apparent when we explain how we answer queries.

To answer an outward query from $q \in P_i$ that is facing both ways, we first project q to the point $\tilde{q}$ along the virtual edge $\tilde{e}$. We distinguish two cases depending on whether $\tilde{q}$ has any farthest bead-string other than B_i or whether B_i is the only farthest bead-string from $\tilde{q}$.

1. Suppose $\tilde{q}$ has some farthest bead-string B_j with $j \neq i$.

   This means that B_j contains outward farthest points from q in A and we can determine the outward farthest distance of q by evaluating the first layer $\mathcal{L}^1$ at $\tilde{q}$, since

   $$\max_{x \in A \setminus B_i} d_A(q, x) = \bar{d}_{P_i \cup B_j}(q) = \Delta_i + \bar{d}_{\tilde{e} \cup B_j}(\tilde{q}) = \Delta_i + \tilde{d}_j(\tilde{q}) = \Delta_i + \mathcal{L}^1(\tilde{q}) \ ,$$

   where the term $\Delta_i = (|P_i| - |P_1|)/2$ corrects the difference between the query from q in A and the projected query from $\tilde{q}$ along the virtual edge $\tilde{e}$ in $\tilde{e} \cup A$. This means that we can calculate the outward farthest-distance from q in $O(\log n)$ time in this case.

   The projected query $\tilde{q}$ stabs those intervals among I^1 that correspond to the functions among $\tilde{d}_1, \tilde{d}_2, \dots, \tilde{d}_\lambda$ that coincide with $\mathcal{L}^1$ at $\tilde{q}$. For every color c with $c \neq i$ such that an interval in I^1 of color c is stabbed by $\tilde{q}$, the bead-string B_c contains an outward farthest point from q, because $\mathcal{L}^1(\tilde{q})$ correctly indicates the outward farthest distance from q.

   Hence, we report any outward farthest points from q as follows. We perform an interval stabbing query in I^1, which takes $O(l + \log n)$ time, where l is the number of reported intervals. If an interval of color i is reported, then we ignore it. For each reported interval of color c with $c \neq i$, we follow the pointer to the edge that contains $\tilde{q}$ in the bead-chain X_c and report all farthest points from q in B_c with a farthest point query from $\tilde{q}$ in X_c. This takes $O(k_c)$ time, where k_c is the number of reported farthest points in B_c. This means that the overall time to report all outward farthest points from q is $O(k + \log n)$.

2. Suppose $\tilde{q}$ has only i as its farthest color.

   This means $\mathcal{L}^1$ only coincides with the function $\tilde{d}_i$ at $\tilde{q}$, i.e., $\tilde{d}_j(\tilde{q}) < \tilde{d}_i(\tilde{q})$ for any $j \neq i$. The value $(|P_i| - |P_1|)/2 + \tilde{d}_i(\tilde{q})$ does not indicate the farthest distance from q to any outward farthest points from q in A. Instead, the outward farthest distance from q is stored with the second highest functions among $\tilde{d}_1, \tilde{d}_2, \ldots, \tilde{d}_\lambda$, i.e., with their second layer $\mathcal{L}^2$.

   This means that we can report the outward farthest distance from q and the outward farthest points from q in almost the same manner as for the previous case: the only difference is that we use the second layer $\mathcal{L}^2$ instead of the first layer $\mathcal{L}^1$ and that we perform interval stabbing queries for the intervals in I^2 instead of the intervals in I^1.

This shows that our construction supports $O(\log n)$-time outward farthest-distance queries and $O(k + \log n)$-time outward farthest-point queries for queries that are facing both ways. This was the last piece of the puzzle to complete the data structure for abacus networks.

Theorem 4.17. *Let A be an abacus with n vertices and λ bead-strings that have σ arcs in total. There is a data structure of size $O(n)$ with $O(n \log \lambda)$ construction time supporting farthest-point queries on N in $O(k + \log n)$ time, where $k \in \{1, 2, \ldots, \lambda + \sigma\}$ is the number of farthest points.*

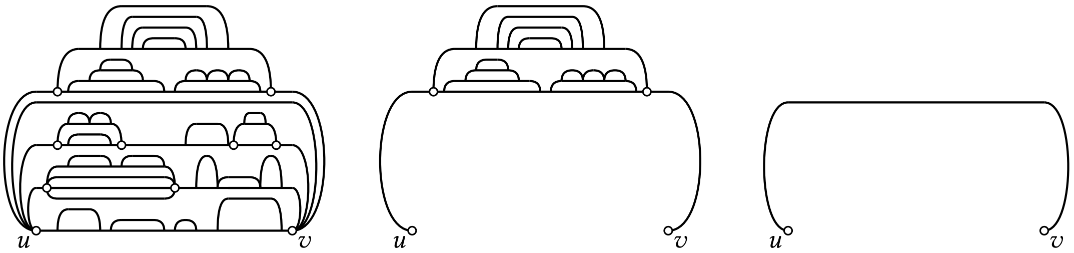
4.5 Two-Terminal Series-Parallel Networks

A two-terminal series-parallel network is generated from a single edge using series and parallel operations. We develop a data structure for queries in bi-connected two-terminal series-parallel networks; the data structure for queries in series-parallel networks from the next section also supports queries in two-terminal series-parallel networks that are not bi-connected.

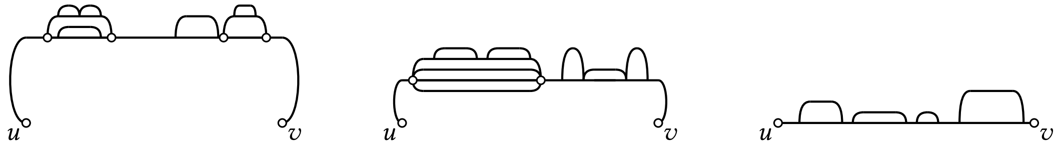
We decompose a bi-connected two-terminal series-parallel network N into a hierarchy of nested two-terminal series-parallel networks. This hierarchy has the shape of a tree where the leaves are two-terminal series-parallel networks that have no further nested structures, i.e., abacus networks, bead-chain networks, and parallel-path networks. We build data structures for queries in the nested networks, recursively, and combine them mimicking the approach for abacus networks. We answer any query with three sub-queries that report the (distance to the) farthest points at the current level of the hierarchy, downward, and upward the hierarchy.

4.5.1 The Nesting Hierarchy

We consider a bi-connected two-terminal series-parallel network N with terminals u and v. A *u-v-component* of N is a maximal sub-network of N that consists of points that can reach each other via a path that contains neither u nor v in its interior. As illustrated in Figure 4.31, u and v are part of each u-v-component of N. Let $K_1, K_2, \ldots, K_\lambda$ be the u-v-components of N.



(a) The network N. (b) The u-v-component K_1. (c) The u-v-component K_2.



(d) The u-v-component K_3. (e) The u-v-component K_4. (f) The u-v-component K_5.

Figure 4.31: A bi-connected two-terminal series-parallel network N with terminals u and v together with its five u-v-components $K_1, K_2, \ldots, K_5$.

The bi-connected components $N_{i,1}, N_{i,2}, \ldots, N_{i,\sigma_i}$ of K_i that are not cycles are the *nested networks* of N that are *attached to* K_i. Each nested network $N_{i,j}$ is itself a bi-connected two-terminal series-parallel network whose terminals are the vertices $a_{i,j}$ and $b_{i,j}$ in $N_{i,j}$ that are closest to u and v in K_i, respectively. The nested networks form a tree that is defined as follows. Recall that parallel-path networks and bead-chain networks are abacus networks by definition. Let N be a bi-connected two-terminal series-parallel network with terminals u and v. If N is an abacus network, then the *nesting hierarchy* $\mathcal{H}$ of N consists of a single node representing N. Otherwise, if N is not an abacus network, then the *nesting hierarchy* $\mathcal{H}$ of N consists of a node representing N whose child nodes are the nesting hierarchies of the nested networks of N. Figure 4.32 illustrates an example of the hierarchy of a two-terminal series-parallel network.

Lemma 4.18. *Every bi-connected two-terminal network has a unique nesting hierarchy.*

Proof. We prove this claim via structural induction. If N is an abacus network, then the nesting hierarchy $\mathcal{H}$ of N consists of a single node and is therefore unique. Suppose N is not an abacus network. If the nested networks of N are unique, then the nesting hierarchy $\mathcal{H}$ of N is unique, since the nesting hierarchies of the nested networks are unique by the induction hypothesis.

We argue that the set of nested networks remains the same even when N is two-terminal with respect to more than one pair of terminals. Suppose that N is two-terminal series parallel with respect to u and v and with respect to u' and v'. If neither u' nor v' lie in a nested network of N with respect to u and v, then the set of nested networks remains the same as well.

Let X be a nested network of N with respect to u and v, and let a and b be the terminals of X that are determined by u and v. Assume, for the sake of a contradiction, that u' lies in X with $a \neq u' \neq b$. Let $P_X(a, u')$ be a shortest path from a to u' in X and let $P_X(u', b)$ be a shortest path from u' to b in X. Since X is bi-connected, there is a path $P_X(a, b)$ in X that connects a and b

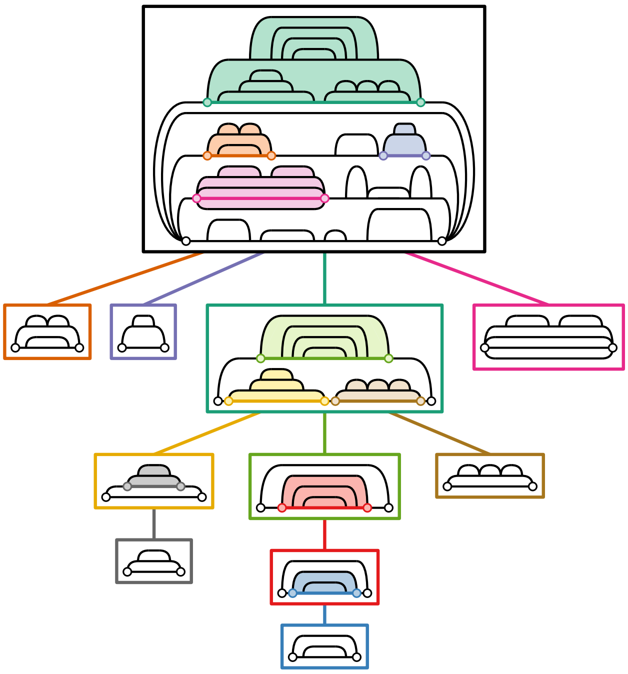


Figure 4.32: The nesting hierarchy of a geometric two-terminal series-parallel network N.

and that neither contains u' nor shares an edge with the path $P_X(a, u') \cup P_X(u', b)$. Furthermore, since N is bi-connected and has terminals u and v, there is a path $P_{N \setminus X}(a, b)$ from a to b outside of X, which contains u and v. As illustrated in Figure 4.33, the vertex u' cannot be a terminal of $N' = P_X(a, b) \cup P_X(a, u') \cup P_X(u', b) \cup P_{N \setminus X}(a, b)$ and, thus, not a terminal of N, since we cannot remove both a and b by reverting series operations and parallel operations in N'.

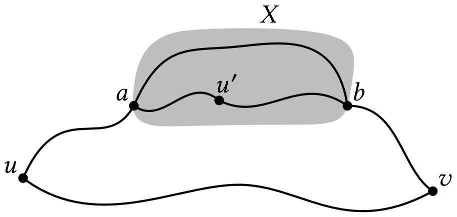


Figure 4.33: The sub-network N' of N from the proof that if u and v are terminals of N and X is a nested network of N with respect to u and v, then N cannot have other terminals u' and v' with $u' \in X$ and $a \neq u' \neq b$. No matter where v' lies, the vertices a and b both cannot be removed by undoing series and parallel operations without first removing u' with a series operation. Therefore, u' cannot be a terminal of N.

Therefore, the set of nested networks remains unique even if the terminals are not unique. This means that the nesting hierarchy $\mathcal{H}$ of N itself is unique, since the nesting hierarchies of the networks $N_1, N_2, \ldots, N_\sigma$ that are nested in N are unique by the induction hypothesis. $\square$

We define the *nesting number* of N as the number τ of nodes in its nesting hierarchy $\mathcal{H}$. For instance, the two-terminal series-parallel network in Figure 4.32 has nesting number $\tau = 11$.

We decompose N into its nested networks $N_1, N_2, \ldots, N_\sigma$ and the abacus A that results from replacing each nested network N_i that is not a cycle with an edge e_i that is weighted with the weighted length of a shortest path in N_i connecting the terminals a_i and b_i of N_i. We refer to A as the *frame* of N. The frame of N is unique, since the nested networks of N are unique. Figure 4.34 illustrates the frames for the hierarchy of the geometric network from Figure 4.32.

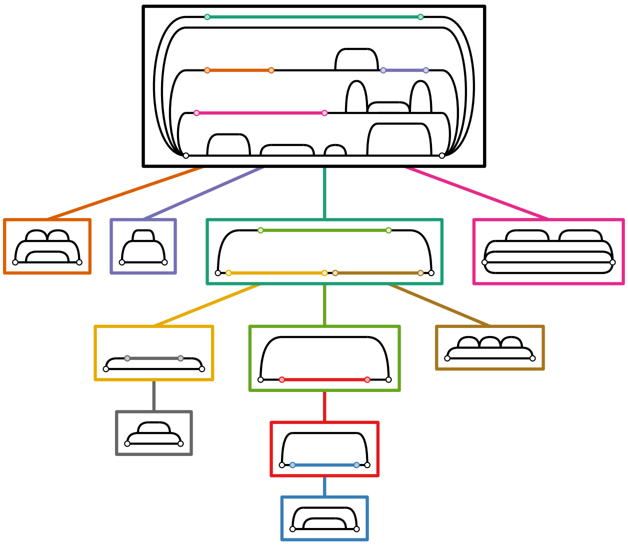


Figure 4.34: The frames of the networks in the nesting hierarchy from Figure 4.32.

Theorem 4.19. *Let N be a bi-connected two-terminal series-parallel network with n vertices that was created using λ parallel operations. It takes $O(n \log \lambda)$ time to construct the nesting hierarchy $\mathcal{H}$ of N alongside with the frames for the networks stored with each node of $\mathcal{H}$.*

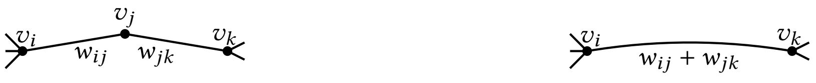
Proof. Our strategy is to revert series operations and parallel operations until we arrive at a single edge connecting two terminals of N while maintaining a log of the operations that were reverted. By reading this log backwards, we obtain a creation history of N alongside with the nesting hierarchy $\mathcal{H}$ as well as the frames of the networks stored at the nodes of $\mathcal{H}$.

We assume that N is represented as follows. The vertices of N are labelled with unique indices $i = 1, 2, \ldots, n$. For each vertex v_i of N, we store a balanced binary search tree B_i that contains the neighbours of v_i sorted by their index. Each entry in B_i refers to an edge of N that is incident to v_i and we store the weight of this edge with the entry of the corresponding neighbour of v_i in B_i. If there are multiple edges connecting the vertices v_i and v_j, then we store a list of the weights of these edges with the entry for v_j in B_i and with the entry for v_i in B_j. Let δ_i be the degree of vertex v_i, for $i = 1, 2, \ldots, n$. With this representation, it takes $O(\log \delta_i + \log \delta_j)$ time to check if two vertices v_i and v_j are adjacent, it takes $O(\log \delta_i + \log \delta_j)$ time to remove a parallel edge connecting v_i and v_j, and it takes $O(\log \delta_i + \log \delta_k)$ time to remove a degree two vertex

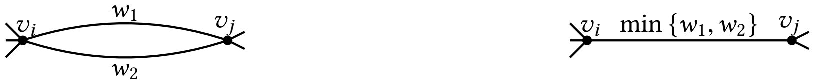
v_j and connect its neighbours v_i and v_j with a new edge. Converting N into this format takes $O\left(n + \sum_{i=1}^{n} \delta_i \log \delta_i\right)$ time, since we sort the indices in the neighbourhood of each vertex.

For each degree $\delta = 1, 2, \ldots, \Delta$, we store a list L_δ of the vertices with degree δ and each vertex of degree δ stores a pointer to its position in L_δ to facilitate constant-time updates of these lists. We maintain a list L_M of the pairs of vertices that are connected by multiple edges. At the beginning, L_M is empty, because N is simple and, thus, has no parallel edges.

We proceed in alternating rounds, where we reverse as many series operations as possible or as many parallel operations as possible in each round, starting with series operations. This means that, at the end of each round, either L_2 is empty, or L_M is empty, or both.



(a) Before reversing a series operation. (b) After reversing a series operation.



(c) Before reversing a parallel operation. (d) After reversing a parallel operation.

Figure 4.35: Reversing series operations and parallel operations.

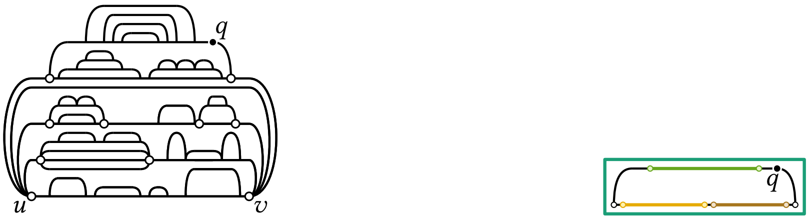
We reverse a series operation as illustrated in Figures 4.35a and 4.35b. Let v_j be a degree-two vertex that is adjacent to the vertices v_i and v_k via edges e_{ij} and e_{jk} of weights w_{ij} and w_{jk}, respectively. We remove v_j from N and introduce a new edge e_{ik} of weight $w_{ik} = w_{ij} + w_{jk}$ that connects v_i and v_k. We update the degree lists by decrementing the degree of v_i and v_j, and we add the pair (i, j) to L_M if v_i and v_j were already connected by an edge. Reversing this series operation and updating the auxiliary lists takes $O(\log \delta_i + \log \delta_k)$ time. We store a record of the series operation that we just reversed, including the indices i, j, and k as well as the weights w_{ij} and w_{jk} that indicate how the addition of v_j subdivides the edge connecting v_i and v_k.

We reverse a parallel operation as illustrated in Figures 4.35c and 4.35d. Let v_i and v_j be two vertices that are connected by parallel edges of weights $|P_1|$ and $|P_2|$. We remove the edge with the smaller weight that connects v_i and v_j and we keep the shorter one. Reversing this series operation and updating the auxiliary lists takes $O(\log \delta_i + \log \delta_j)$ time. We store a record of the parallel operation that we just reversed alongside with the weight of the deleted edge.

For every two-terminal series-parallel network, this procedure terminates with a single edge that connects the two terminals u and v of N [20]. The weight of this final edge is the length of a shortest path in N that connects its terminals u and v. Reading the record of reversed operations backwards yields a creation history for N. We obtain the nesting hierarchy $\mathcal{H}$ of N together with the frames for each node of $\mathcal{H}$ by tracing this creation history as follows.

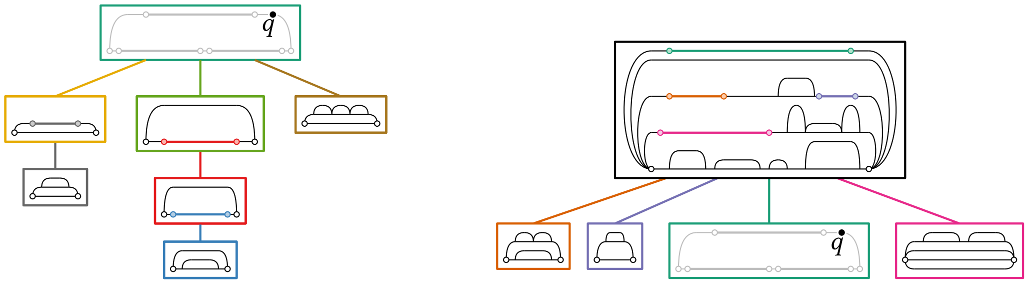
We redo the series operations and the parallel operations in the creation history of N starting with a single edge connecting the terminals u and v. We initialize the nesting hierarchy $\mathcal{H}$ as a single node that stores the edge connecting u and v as its initial frame. When we redo a series operation that subdivides an edge e, the nesting hierarchy remains unchanged and we update the frame that contains e by subdividing e according to the recorded weights. When we redo a parallel operation, we tentatively create a new node in the nesting hierarchy and store with it a new frame consisting of the two edges that were involved in the parallel operation. This may create extraneous nodes in the temporary nesting hierarchy that contain only cycles. Once we have reversed all operations, we merge the leaves of the nesting hierarchy that contain only cycles as frames back into their parent nodes to obtain the final hierarchy with frames.

Whenever we create a new nested network, the edge to which the parallel operation was applied had the weight of a shortest path through the nested network that is about to be generated. Therefore, the networks stored with the nodes of the nesting hierarchy are indeed the frames of the network at any time throuhgout this process. We can perform the above construction in $O\left(\sum_{i=1}^{n} \delta_i \log \delta_i\right)$ time, if each edge stores its current containing frame. The maximum degree in N is at most $\lambda + 2$, where λ is the number of parallel operations required to generate N, i.e., $\log \delta_i = O(\log \lambda)$ for each $i = 1, 2, \ldots, n$. Furthermore, $\sum_{i=1}^{n} \delta_i = O(n)$, since N is planar [20]. Therefore, the overall construction time for $\mathcal{H}$ and its associated frames is $O(n \log \lambda)$. □



(a) A query q in N.

(b) The local query for q.



(c) The downward query for q.

(d) The upward query for q.

Figure 4.36: An illustration of (a) some query q in the network N from Figure 4.31 together with (b) the local query that reports farthest points from q in the frame of N that contains q, and (c) the downward query that reports farthest points from q that we reach by descending the nesting hierarchy $\mathcal{H}$ of N, and (d) the upward query that reports farthest points from q that we reach by ascending the nesting hierarchy.

We use the nesting hierarchy $\mathcal{H}$ of N to decompose a query from $q \in N$ into three sub-queries: a *local query* for the (distance to the) farthest points in the frame A of the hierarchy $\mathcal{H}$ containing q, a *downward query* for the (distance to the) farthest points from q to any point that we reach by descending $\mathcal{H}$ from A, and an *upward query* for the (distance to the) farthest points from q to any point that we reach by ascending $\mathcal{H}$ from A. Figure 4.36 illustrates the parts of the network and the nesting hierarchy that correspond to each of these three sub-queries for a query point.

4.5.2 Local Query

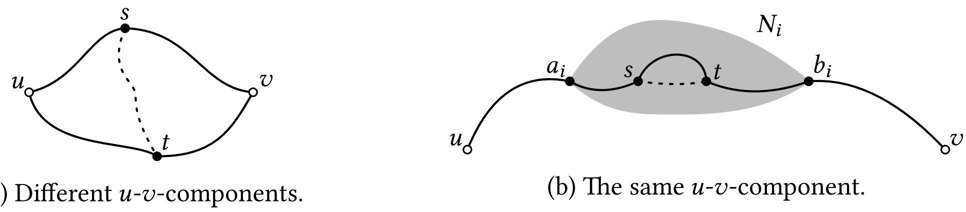
We begin with local queries in the frame that corresponds to the root of the nesting hierarchy. Let A be the frame of a two-terminal series-parallel network N and let u and v be the terminals of N. We define a sub-network F of N to translate between the distances in A and in N. Let $K_1, K_2, \dots, K_\lambda$ be the u-v-components of N, and let $N_1, N_2, \dots, N_\sigma$ be the nested networks of N. Recall that the abacus network A results from replacing each nested network N_i with terminals a_i and b_i by a dummy edge e_i of length $d_{N_i}(a_i, b_i)$. Let F be the network that results from replacing each edge e_i with a shortest path π_i from a_i to b_i in N_i for each $i = 1, 2, \dots, \lambda$.

We define local queries in A more precisely. Consider a query point $q \in N$ with $q \in A$. A local farthest-distance query from q in A reports the farthest distance from q along F in N, i.e., $\max_{x \in F} d_N(q, x)$; we argue that this value is $\bar{d}_A(q)$. A local farthest-point query from q in A reports the farthest points from q in F that lie in A; we argue that these are the farthest points from q in A that lie in N. We show that the network distance of F and N are compatible.

Lemma 4.20. *For every $p, q \in F$, we have $d_F(p, q) = d_N(p, q)$.*

Proof. For every pair $p, q \in N$, we have $d_F(p, q) \geq d_N(p, q)$, because F is a sub-network of N. We show that every shortest path in F is a shortest path in N and, thus, $d_F(p, q) \leq d_N(p, q)$.

We consider a shortest path $P_F(p, q)$ that connects p with q in F. Assume, for the sake of a contradiction, that some shortest path $P_N(p, q)$ that connects p with q in N is strictly shorter than $P_F(p, q)$. There exist two distinct points $s, t \in P_F(p, q) \cap P_N(p, q)$ such that the sub-path $P_N(s, t)$ from s to t along $P_N(p, q)$ is strictly shorter than the sub-path $P_F(s, t)$ from s to t along F and the interior of the path $P_N(s, t)$ does not contain any points from F. Note that neither u nor v lie on $P_N(s, t)$, since $u, v \in F$. We distinguish two cases depending on whether s and t lie in the same u-v-components of N or not and we derive a contradiction in each case.



(a) Different u-v-components.

(b) The same u-v-component.

Figure 4.37: The two impossible locations for s and t with the path $P_N(s, t)$ shown dashed.

1. Suppose s and t lie in different u-v-components of N.

   Let K_s and K_t be the u-v-components of N with $s \in K_s$ and $t \in K_t$. Then, N contains the diamond in Figure 4.37a formed by the path from u to v in K_s via s, the path from u to v in K_t via t and the path $P_N(s, t)$. This is a contradiction, since this diamond cannot occur in a network that is two-terminal with respect to u and v.

2. Suppose s and t lie in the same u-v-component K of N.

   Recall that F contains a shortest path from u to v in K and all bi-connected components of K that are cycles. As the interior of $P_N(s,t)$ lies outside of F, the entire path $P_N(s,t)$ lies in some nested network N_i of N that is part of K, as illustrated in Figure 4.37b.

   Recall that $F \cap N_i$ is a shortest path π_i in N_i that connects the terminals a_i and b_i of N_i. We arrive at a contradiction, since s and t must lie on π_i. Yet, replacing $P_F(s,t)$ with $P_N(s,t)$ in π_i yields a strictly shorter path from a_i to b_i in N_i. This is impossible.

Since both cases lead to a contradiction s and t—and, thus, $P_N(p,q)$—cannot exist. Hence, every shortest path in F is also a shortest path in N and $d_F(p,q) = d_N(p,q)$ for every $p,q \in F$. □

Lemma 4.20 implies that the answer to a local farthest-distance query from $q \in F$ is

$$\max_{x \in F} d_N(q,x) = \max_{x \in F} d_F(q,x) = \bar{d}_F(q) \ .$$

We argue that $\bar{d}_F(q) = \bar{d}_A(q)$. Let $\phi : A \to F$ be the bijection that is defined as follows. For $p \in A$ with $p \in N$, let $\phi(p) = p$. For $p \in A$ with $p \in e_i$ for $i = 1,2,\ldots,\sigma$, let $\phi(p)$ be the unique point along π_i with $d_{e_i}(a_i,p) = d_{\pi_i}(a_i,\phi(p))$. We have $d_A(p,q) = d_F(\phi(p),\phi(q))$ for every $p,q \in A$. Therefore, for every query $q \in N$ with $q \in A$, we observe $\bar{d}_A(q) = \bar{d}_F(q)$, since

$$\bar{d}_A(q) = \max_{p \in A} d_A(q,p) = \max_{\phi(p) \in F} d_F(\phi(q),\phi(p)) = \max_{\phi(p) \in F} d_F(q,\phi(p)) = \bar{d}_F(q) \ .$$

This means we can answer a local farthest-distance query directly with a query in A. Therefore, a local farthest-distance query in A takes $O(\log n)$ time after $O(n \log \lambda)$ pre-processing.

If $p \in A$ is a farthest point from q in A, then $\phi(p)$ is a farthest point from q in F, since

$$\bar{d}_F(q) = \bar{d}_A(q) = d_A(q,\bar{q}) = d_F(q,\phi(\bar{q})) \ ,$$

and if $\phi(p)$ is a farthest point from q in F, then p is a farthest point from q in A, since

$$\bar{d}_A(q) = \bar{d}_F(q) = d_F(q,\phi(p)) = d_A(q,p) \ .$$

If there are farthest points from q in N along F, then we report them with a query from q in A. This query might return farthest points from q in A that lie along the dummy edges $e_1, e_2, \ldots, e_\sigma$; we remove these artifacts from the output. Each farthest point p from q in A on a dummy edge corresponds to a farthest point $\phi(p)$ from q in N. Therefore, a local farthest-point query from q in A requires $O(k + \log n)$ time, where k is the number of farthest points from q in N.

So far, we have described how to support local queries for the frame A of N itself, i.e., the frame that corresponds to the root node of the nesting hierarchy $\mathcal{H}$ of N. Let A_i be the frame of the nested network N_i of N and let a_i and b_i be the terminals of N_i with respect to $\mathcal{H}$. When we apply the above technique to support local queries in A_i, we need to account for potential shortest paths between points in A_i that leave N_i. These shortest paths exist precisely when $d_{N_i}(a_i,b_i) > d_N(a_i,b_i)$. In this case, we support local queries in A_i with the data structure in the abacus network $A_i \cup e'$, where e' is an additional edge of length $d_N(a_i,b_i)$. Otherwise, we support local queries in A_i with the data structure in the abacus network A_i itself.

Lemma 4.21. *Let N be a bi-connected two-terminal series-parallel network with n vertices and parallelism λ. There exists a data structure with $O(n \log \lambda)$ construction time and $O(n)$ size that supports $O(\log n)$-time local farthest-distance queries and $O(k + \log n)$-time local farthest-point queries for all frames associated with N, where k is the number of reported local farthest points.* □

4.5.3 Downward Query

We construct the data structure for downward queries with a bottom-up approach starting from the leaves of the nesting hierarchy towards the root. We describe how to perform downward queries from the root of the nesting hierarchy assuming that the data structures for the remaining nodes have already been created. The procedure is analogous for nodes at lower levels.

We consider a bi-connected two-terminal series-parallel network N with terminals u and v. Let A be the frame of N, let $K_1, K_2, \ldots, K_\lambda$ be the u-v-components of N, and for $i = 1, 2, \ldots, \lambda$, let $N_{i,1}, N_{i,2}, \ldots, N_{i,\sigma_i}$ be the nested networks of N in K_i. A *downward farthest-distance query* from $q \in A$ reports the farthest distance from q in N to any point on a nested network, i.e.,

$$\max_{i=1}^{\lambda} \max_{j=1}^{\sigma_i} \max_{p \in N_{i,j}} d_N(p, q) \ ,$$

and a *downward farthest-point query* from q reports all farthest points from q in N that lie in any of the nested networks of N. We mimic the approach for farthest-arc queries in an abacus network and divide a downward query $q \in A$ with $q \in K_i$ into two sub-queries: An *inward downward query* from q reports the (distance to the) farthest points from q with respect to N that lie in the nested networks along the u-v-component K_i that contains q. An *outward downward query* from q that reports the (distance to the) farthest points from q with respect to N that lie in the nested networks along the other u-v-components K_j with $j \neq i$.

Inward Downward Queries

To support inward downward queries in the u-v-component K_i, we need to take into account that the shortest path from $q \in A \cap K_i$ to point $x \in N_{i,j}$ may leave and reenter K_i through the terminals u and v. We consider the network K_i' that consists of K_i and an edge e_i' of length $d_{N \setminus K_i}(u, v)$. For every pair $p, q \in K_i$, we have $d_N(p, q) = d_{K_i'}(p, q)$ and, thus, $\bar{d}_{K_i'}(q) = \max_{p \in K_i} d_N(p, q)$.

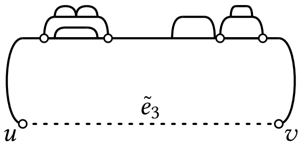


Figure 4.38: The network K_3' for the network from Figure 4.31.

If A contains the u-v-paths $P_1, P_2, \ldots, P_\lambda$ of lengths $|P_1| \leq |P_2| \leq \cdots \leq |P_\lambda|$, then we have $d_{N \setminus K_i}(u, v) = |P_1|$ for $i = 2, 3, \ldots, \lambda$ and $d_{N \setminus K_i}(u, v) = |P_2|$ for $i = 1$. This means we obtain the length $d_{N \setminus K_i}(u, v)$ of the edge e_i' in K_i' as a by-product of the construction of A.

Let N_j be a nested network of N with terminals a_j and b_j that belongs to K_i. In A, the nested network N_j is replaced by an edge e_j of length $d_{N_j}(a_i, b_i)$ that corresponds to a shortest path π_j in N_j from a_j to b_j. We identify $p \in e_j$ with the unique point $\pi_j(p)$ along π_j such that $d_{e_j}(a_j, p) = d_{\pi_j}(a_j, \pi_j(p))$.

Suppose we already have a data structure for queries in N_j with a virtual edge $\tilde{e}_j$ of length $d_{N_j}(a_j, b_j)$ for outward queries in the frame of N_j and downward outward queries from N_j.

Consider the cycle C_i formed by P_i and the edge e_i' in K_i' that represents the shortest path from u to v outside of the u-v-component K_i. Without loss of generality, we consider only inward downward queries from the cycle C_i. Similarly to queries from the arcs of a bead-chain, we answer inward downward queries from the arcs of K_i' with a corresponding query from C_i. A nested network N_j is *good*, if $d_{N_j}(a_j, b_j) \leq d_N(a_j, b_j)$, i.e., $d_{N_j}(a_j, b_j) \leq |C_i|/2$ and N_j is *bad*, otherwise. There is at most one *bad* nested network in K_i and we treat it like a bad arc.

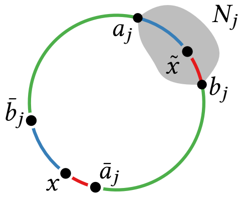


Figure 4.39: The points along C_i.

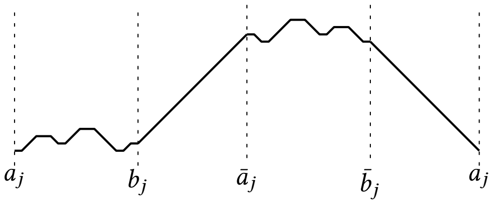


Figure 4.40: The shape of the function $\hat{d}_j(x)$ for $x \in C_i$.

Let $\bar{a}_j$ and $\bar{b}_j$ be the farthest point from a_j and b_j along C_i, respectively. When we traverse the cycle C_i, we encounter the points b_j, $\bar{a}_j$, $\bar{b}_j$, a_j in this order or its reverse, as in Figure 4.39. Note that we have $d_{C_i}(\bar{a}_j, \bar{b}_j) = d_{C_i}(a_j, b_j)$ and $d_{C_i}(a_j, \bar{b}_j) = d_{C_i}(\bar{a}_j, b_j)$. For $x \in C_i$ between $\bar{b}_j$ and $\bar{a}_j$, let $\tilde{x}$ be the unique point on the virtual edge $\tilde{e}_j$ of N_j with $d_{C_i}(x, \bar{a}_j) = d_{\tilde{e}_j}(\tilde{x}, b_j)$.

We describe the farthest distance $\hat{d}_j(x)$ from points $x \in C_i$ to any point along N_j in K_i', i.e.,

$$\hat{d}_j(x) = \max_{y \in N_j} d_{K_i'}(x, y) = \max_{y \in N_j} d_N(x, y) \quad .$$

When x lies between a_j and b_j, the farthest points from x along N_j in K_i' are the farthest points from $\pi_j(x)$ in N_j. When x lies between b_j and $\bar{a}_j$, the farthest points from x along N_j in K_i' are the farthest points from b_j in N_j. When x lies between $\bar{a}_j$ and $\bar{b}_j$, the farthest points from x along N_j in K_i' are the farthest points from $\tilde{x}$ in $N_j \cup \tilde{e}_j$. When x lies between $\bar{b}_j$ and a_j, the farthest points from x in N_j are the farthest points from a_j in N_j. For $x \in C_i$, this yields

$$\hat{d}_j(x) = \begin{cases} \bar{d}_{N_j}(\pi_j(x)) & \text{, if } x \text{ lies between } a_j \text{ and } b_j \\ d_{C_i}(x, b_j) + \bar{d}_{N_j}(b_j) & \text{, if } x \text{ lies between } b_j \text{ and } \bar{a}_j \\ d_{C_i}(b_j, \bar{a}_j) + \bar{d}_{N_j \cup \tilde{e}_j}(\tilde{x}) & \text{, if } x \text{ lies between } \bar{a}_j \text{ and } \bar{b}_j \\ d_{C_i}(x, a_j) + \bar{d}_{N_j}(a_j) & \text{, if } x \text{ lies between } \bar{b}_j \text{ and } a_j \end{cases} \quad .$$

As illustrated in Figure 4.40, the plot of the function $\hat{d}_j$ consists of an increasing line segment of slope plus one from b_j to $\bar{a}_j$, a high rugged plateau in form of the plot of the function $\bar{d}_{N_j \cup \tilde{e}_j}(\tilde{x})$ elevated by $d_{C_i}(b_j, \bar{a}_j)$ from $\bar{a}_i$ to $\bar{b}_j$, a decreasing line segment of slope minus one from $\bar{b}_j$ to a_j, and a low rugged plateau in form of the plot of the function $\bar{d}_{N_j}(\pi_j(x))$ from a_j to b_j.

We consider the upper envelope $\hat{D}_i$ of the functions $\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_{\sigma_i}$ that correspond to the nested networks $N_1, N_2, \ldots, N_{\sigma_i}$ of N that are part of K_i, i.e., $\hat{D}_i(x) = \max_{j=1}^{\sigma_i} \hat{d}_j(x)$ for $x \in C_i$. For a query $q \in K_i' \cap A$, the downward inward farthest distance from q is $\hat{D}_i(q)$, since

$$\hat{D}_i(q) = \max_{j=1}^{\sigma_i} \hat{d}_j(q) = \max_{j=1}^{\sigma_i} \max_{p \in N_j} d_{K_i'}(p, q) = \max_{j=1}^{\sigma_i} \max_{p \in N_j} d_N(p, q) \quad .$$

Similar to the arc distance functions for the good arcs of a bead-chain network, for the good nested networks, the rugged high plateaus do not overlap, the rugged low plateaus do not overlap, and the cyclic order of each type of plateau matches the cyclic order of the corresponding nested networks along C_i. This means that we can compute $\hat{D}_i$ with the technique from Lemma 4.6 in $O(n_i)$ time, where n_i is the number of vertices in K_i. Therefore, we can support $O(\log n)$-time inward downward farthest-distance queries in all u-v-components of N after $O(n)$ preprocessing, provided that this construction has already been completed for all nested networks of N.

We discuss how to support inward downward farthest-point queries from $q \in K_i' \cap A$. Let N_j be a good nested network of N in K_i with terminals a_j and b_j, and let x be some query point along C_i in K_i'. If x lies between b_j and $\bar{a}_j$, then the farthest points from x in N_j are the farthest points from b_j in N_j. If there are k_j farthest points from x in N_j, then we can report them in $O(k_j)$ time, provided that we store them with the data structure for queries in N_j. Likewise, we report the farthest points from x in N_j when x lies between $\bar{b}_j$ and a_j. When x lies between a_j and b_j, the farthest points from x in N_j are the farthest points from $\pi_j(x)$ in N_j; we report them using a window list W_j for farthest-point queries in N_j that was stored with the function $\bar{d}_{N_j}(p)$ for $p \in \pi_j$. For each edge of $\hat{D}_i$ along the low rugged plateau of $\hat{d}_j$, we store a pointer to the corresponding window in W_j. Likewise, each rugged plateau of $\hat{D}_i$ along the high rugged plateau of $\hat{d}_j$ stores a pointer to the corresponding window in the window list $\tilde{W}_j$ for farthest point queries along the virtual edge $\tilde{e}_j$ of N_j. We create these pointers when constructing $\hat{D}_i$ alongside with a window list W_i to support queries for the nested networks that determine $\hat{D}_i$ for any query point along C_i. With this preparation, it takes $O(k + \log n_i)$ time to report all inward farthest points from $q \in K_i' \cap A$, where k is the number of reported points.

We refer to the depth δ of the nesting hierarchy $\mathcal{H}$ of N as the *nesting depth* of N. The total time to construct all data structures for inward downward queries in the nesting hierarchy $\mathcal{H}$ amounts to $O(\delta n \log \lambda)$ and the total size of these data structures amounts to $O(\delta n)$, because every edge e of N may contribute a constant number of times to each data structure at the nodes along a path from the root of $\mathcal{H}$ to the node whose frame contains the edge e.

Lemma 4.22. *Let N be a bi-connected two-terminal series-parallel network with n vertices, parallelism λ, and nesting depth δ. There exists a data structure with $O(\delta n \log \lambda)$ construction time and $O(\delta n)$ size that supports $O(\log n)$-time inward downward farthest-distance queries and $O(k+\log n)$-time inward downward farthest-point queries for all frames associated with the nesting hierarchy $\mathcal{H}$ of N, where k is the number of reported inward downward farthest points.* □

Outward Downward Queries

Let $N_{i,1}, N_{i,2}, \ldots, N_{i,\sigma_i}$ be the nested networks of N in the u-v-component K_i of N for each $i = 1, 2, \ldots, \lambda$. An outward downward farthest-distance query from $q \in K_i \cap A$ reports the farthest distance from q in N to the nested networks of u-v-components K_j with $j \neq i$, i.e.,

$$\max_{j=1, j\neq i}^{\lambda} \max_{l=1}^{\sigma_j} \max_{p \in N_{j,l}} d_N(p, q) \ ,$$

and an outward farthest-point query from $q \in K_i \cap A$ reports the farthest points from q in N that lie on a nested network $N_{j,l}$ with $j \neq i$ and $l = 1, 2, \ldots, \sigma_l$, as illustrated in Figure 4.41.

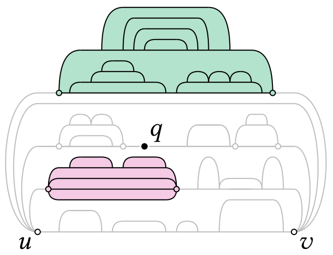


Figure 4.41: An outward downward query from $q \in K_3 \cap A$ where we consider farthest points from q in the coloured nested networks.

We mimic our approach for outward queries in an abacus network. The query q is *facing* u when every shortest path from q to v passes through u, the query q is *facing* v when every shortest path from q to u passes through v, and the query q is *facing both ways*, otherwise.

When q is facing u, then the outward downward farthest points from q are the farthest points from u on any nested network $N_{j,l}$ with $j \neq i$, i.e., the outward farthest distance from q is

$$d_{K_i}(q,u) + \max_{j=1, j\neq i}^{\lambda} \max_{l=1}^{\sigma_j} \max_{p \in N_{j,l}} d_N(p,u) \ .$$

To support u-facing outward downward queries, we determine the farthest distance from u to the nested networks in each u-v-component, i.e., we determine $o_j = \max_{l=1}^{\sigma_j} \max_{p \in N_{j,l}} d_N(p,u)$ for each $j = 1, 2, \ldots, \lambda$. Let $j_1, j_2, \ldots, j_\lambda$ be a reordering of $1, 2, \ldots, \lambda$ such that $o_{j_1} \leq o_{j_2} \leq \cdots \leq o_{j_\lambda}$. The outward farthest distance from a u-facing query $q \in K_i \cap A$ is

$$d_{K_i}(q,u) + \max_{j=1, j\neq i}^{\lambda} \max_{l=1}^{\sigma_j} \max_{p \in N_{j,l}} d_N(p,u) = d_{K_i}(q,u) + \max_{j=1, j\neq i}^{\lambda} o_j = d_{K_i}(q,u) + \begin{cases} o_{j_\lambda} & \text{, if } i \neq j_\lambda \\ o_{j_{\lambda-1}} & \text{, if } i = j_\lambda \end{cases} \ .$$

Thus, answering u-facing outward farthest-distance queries takes $O(1)$ time. We obtain $o_{j_{\lambda-1}}$ and o_{j_λ} in $O(n)$ time when constructing the data structures for inward downward queries.

Let O_j be the set of farthest points from u in the nested networks of K_j, i.e., let

$$O_j = \bigcup_{l=1}^{\sigma_j} \{ p \in N_{j,l} \mid d_N(p,u) = o_j \} \ .$$

For a query from $q \in K_i \cap A$ that is facing u, the outward downward farthest points from q are stored in each O_j with $o_j = o_{j_\lambda}$ when $i \neq j_\lambda$, and in each O_j with $o_j = o_{j_{\lambda-1}}$ when $i = j_\lambda$. Therefore, we support $O(k)$-time u-facing outward downward farthest-point queries, where k is the number of reported points, by determining the sets O_{j_λ} and O_j with $o_j = o_{j_{\lambda-1}}$ in $O(n)$ time when constructing the data structures for inward downward farthest-point queries.

We generalize the construction for outward queries that are facing both ways from abacus networks to outward downward queries that are facing both ways in a two-terminal network. Let $P_1, P_2, \ldots, P_\lambda$ be the u-v-paths in A of lengths $|P_1| \leq |P_2| \leq \cdots \leq |P_\lambda|$, where P_i is a shortest path from u to v in the u-v-component K_i for each $i = 1, 2, \ldots, \lambda$. In the following, we only discuss outward downward queries that are facing both ways from query points $q \in K_i \cap A$ along the u-v-path P_i. We translate any queries from the arcs of A in K_i to a query from P_i.

We introduce a virtual edge $\tilde{e}$ of length $d_N(u,v) = |P_1|$ that connects the terminals u and v. For $i = 1, 2, \ldots, \lambda$, let $\tilde{d}_i$ denote the function that maps the points $\tilde{q}$ on the virtual edge $\tilde{e}$ to their farthest distance to any point on any nested network of K_i with respect to $K_i \cup \tilde{e}$, i.e.,

$$\tilde{d}_i(\tilde{q}) = \max_{l=1}^{\sigma_i} \max_{p \in N_{i,l}} d_{K_i \cup \tilde{e}}(p,q) = \max_{l=1}^{\sigma_j} \max_{p \in N_{j,l}} d_N(p,q) \ .$$

We obtain $\tilde{d}_i$ when constructing the data structure for inward downward queries in K_i, because $\tilde{d}_i$ is the restriction of the downward farthest distance function $\hat{D}_i$ from K'_i to the edge e'_i. Following the approach for abacus networks, we determine the first layer $\mathcal{L}^1$ and the second layer $\mathcal{L}^2$ of $\tilde{d}_1, \tilde{d}_2, \ldots, \tilde{d}_\lambda$ with interval stabbing data structures that report which of the functions determine each layer at any query point. This construction takes $O(n \log \lambda)$ time, provided that we have already completed this construction for descendents of N in the nesting hierarchy.

When answering an outward downward farthest-distance query from $q \in K_i \cap A$ that is facing both ways, we project q onto the unique point $\tilde{q}$ along $\tilde{e}$ with $d_N(\bar{v}_i, q) = d_{\tilde{e}}(u, \tilde{q})$ and $d_N(q, \bar{u}_i) = d_{\tilde{e}}(\tilde{q}, v)$, where $\bar{v}_i$ and $\bar{u}_i$ are the farthest point from v and u on P_i in N, respectively. We call $\tilde{q}$ the *projection* of q onto $\tilde{e}$. If $\tilde{q}$ has a farthest point on some $N_{m,l}$ with $m \neq i$ in $N \cup \tilde{e}$, then $\tilde{d}_m(\tilde{q}) = \mathcal{L}^1(\tilde{q})$ and we can read the outward farthest distance from q from $\mathcal{L}^1$, since

$$\max_{j=1, j\neq i}^{\lambda} \max_{l=1}^{\sigma_j} \max_{p \in N_{j,l}} d_N(p, q) = \frac{|P_i| - |P_1|}{2} + \max_{j=1, j\neq i}^{\lambda} \tilde{d}_j(\tilde{q}) = \frac{|P_i| - |P_1|}{2} + \mathcal{L}^1(\tilde{q}) \ ,$$

where the term $(|P_i| - |P_1|)/2$ corrects the projection of the query from the u-v-path P_i of length $|P_i|$ to the virtual edge $\tilde{e}$ of length $|P_1| = d_N(u, v)$. If $\tilde{q}$ has all downward farthest points on K_i itself, then we can read the outward farthest distance from q from $\mathcal{L}^2$, since

$$\max_{j=1, j\neq i}^{\lambda} \max_{l=1}^{\sigma_j} \max_{p \in N_{j,l}} d_N(p, q) = \frac{|P_i| - |P_1|}{2} + \max_{j=1, j\neq i}^{\lambda} \tilde{d}_j(\tilde{q}) = \frac{|P_i| - |P_1|}{2} + \mathcal{L}^2(\tilde{q}) \ .$$

Therefore, answering an outward downward farthest-distance query takes $O(\log n)$ time, since we can use the interval stabbing data structures to decide in $O(\log n)$ time if $\tilde{q}$ has farthest points on any u-v-component other than K_i itself and evaluating $\mathcal{L}^1$ or $\mathcal{L}^2$ takes $O(\log n)$ time.

We wish to report the outward downward farthest points from $q \in K_i$, i.e., the farthest points from q on any u-v-components K_j with $j \neq i$. We project q to $\tilde{q}$ onto the virtual edge $\tilde{e}$ and use the interval stabbing data structure for the first layer to check if $\tilde{q}$ has any farthest points on any u-v-component other than K_i itself. If this is the case, we proceed to report the farthest points from all u-v-components that were reported from the interval stabbing query. For each reported u-v-component K_j, we project the query from $\tilde{q}$ to the virtual edge of K'_j, i.e., the data structure for inward farthest-point queries in K_j. The query in K_j takes $O(k_j)$ time to answer, where k_j is the number of reported farthest points from q in K_j, if we break the intervals in the interval stabbing data structure depending on the answer to the query in K'_j. Likewise, we proceed with the second layer, if all farthest points from $\tilde{q}$ in $N \cup \tilde{e}$ lie along K_i itself. Therefore, it takes $O(k + \log n)$ time to answer an outward downward farthest-point query.

The total time to construct all data structures for outward downward queries in the nesting hierarchy $\mathcal{H}$ amounts to $O(\delta n \log \lambda)$ and the total size of these data structures amounts to $O(\delta n)$, because every edge e of N may contribute a constant number of times to each data structure at the nodes along a path from the root of $\mathcal{H}$ to the node whose frame contains the edge e.

Lemma 4.23. *Let N be a bi-connected two-terminal series-parallel network with n vertices, parallelism λ, and nesting depth δ. There exists a data structure with $O(\delta n \log \lambda)$ construction time and $O(\delta n)$ size that supports $O(\log n)$-time outward downward farthest-distance queries and $O(k + \log n)$-time outward downward farthest-point queries for all frames associated with the nesting hierarchy $\mathcal{H}$ of N, where k is the number of reported outward downward farthest points.* □

4.5.4 Query Upward

We support upward queries by propagating information from the root of the nesting hierarchy back to its leaves after building the data structures for local queries and downward queries.

Let N_j be a nested network of N at the u-v-component K_i of N. Furthermore, let a_j and b_j be the terminals of N_j and let A_j be the frame of N_j. We consider an upward query from $q \in A_j$, i.e., we would like to determine the farthest distance from q in N to any point on $N \setminus N_j$ and we would like to report any farthest points from q in N that lie in $N \setminus N_j$. To support these queries, we translate the query from q to a query from the edge e_j that replaces N_j in the frame A of N. Then, we gather all information about (the distance to) farthest points in $N \setminus N_j$ along e_j.

We distinguish the three familiar cases. The query q is *facing* a_j when all shortest paths in N from q to b_j pass through a_j, the query q is *facing* b_j when all shortest paths in N from q to a_j are passing through b_j, and the query q is *facing both ways*, otherwise.

When q is facing a_j, then all shortest paths from q to any point on $N \setminus N_j$ pass through a_j. Therefore, the farthest distance from $q \in A_j \cap N_j$ to any point on $N \setminus N_j$ is

$$\max_{x \in N \setminus N_j} d_N(q, x) = d_{N_j}(q, a_j) + \max_{x \in N \setminus N_j} d_N(a_j, x) = d_{A_j}(q, a_j) + \bar{d}_{N \setminus N_j}(a_j) \ .$$

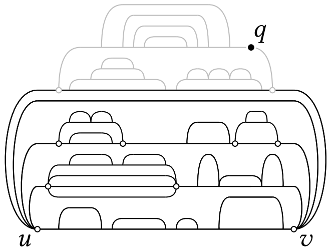


Figure 4.42: An upward query.

We obtain the value $\bar{d}_{N \setminus N_j}(a_j)$ and the farthest points from a_j in $N \setminus N_j$ by combining a query from a_j in the frame A of N and a downward query from a_j in $N \setminus N_j$. We modify the data structures for inward downward queries in N to answer the inward downward query from a_j in $N \setminus N_j$. If a_j has any inward downward farthest point from a_j in N that lies in $K_i \setminus N_j$, then the inward farthest-distance query from a_j in K_i' reports the inward farthest distance from q in $N \setminus N_j$. Otherwise, the inward downward farthest distance from a_j in $N \setminus N_j$ is the height of the second layer $\hat{D}_i^2$ of the functions $\hat{d}_1, \hat{d}_2, \dots, \hat{d}_{\sigma_i}$ at q. We add the construction of $\hat{D}_i^2$ as an additional step when building the data structures for inward downward queries. The data structure for outward downward queries already supports the query for a_j in $N \setminus N_j$, since it reports the (distance to the) downward farthest points from a_j on $N \setminus K_i$. Therefore, for upward queries from N_j that are facing a_j, we can report the upward farthest distance from q in $O(1)$-time and we can report all k upward farthest points from q in $O(k)$ time by storing the farthest distance from a_j in $N \setminus N_j$ and the farthest points from a_j in N that lie on $N \setminus N_j$. This requires $O(n)$ time and space for the root of $\mathcal{H}$, provided that the modified data structures for inward downward queries in the nested networks of N have already been constructed.

Consider an upward query from a point $q \in A_j \cap N_j$ that is facing both ways. Recall that e_j denotes the edge of A that represents a shortest path π_j from a_j to b_j in N_j. We project q onto the unique point q' along e_j with the same relative position to a_j and b_j as q in N_j, i.e.,

$$\frac{d_{N_j}(a_j, q)}{d_{N_j}(a_j, q) + d_{N_j}(q, b_j)} = \frac{d_{e_j}(a_j, q')}{d_{e_j}(a_j, q') + d_{e_j}(q', b_j)} \ .$$

This generalizes the translation of a query from an arc of an abacus network to a query on its parallel-path network. The upward farthest distance from $q \in A_j \cap N_j$ in N is

$$\max_{x \in N \setminus N_j} d_N(q, x) = \frac{d_{N_j}(a_j, q) + d_{N_j}(q, b_j) - d_{N_j}(a_j, b_j)}{2} + \bar{d}_{N \setminus N_j}(q') \ ,$$

where the term $(d_{N_j}(a_j, q) + d_{N_j}(q, b_j) - d_{N_j}(a_j, b_j))/2$ corrects the projection from N_j to A. Computing the correcting term takes constant time, if each vertex of A_j stores its distance to the terminals a_j and b_j. As described for queries facing a_j, we can answer the query for $\bar{d}_{N \setminus N_j}(q')$ in $O(\log n)$ time using the data structures for local queries in A and for downward queries from N along e_j and we can report any farthest points from q' along $N \setminus N_j$ in $O(k + \log n)$ time.

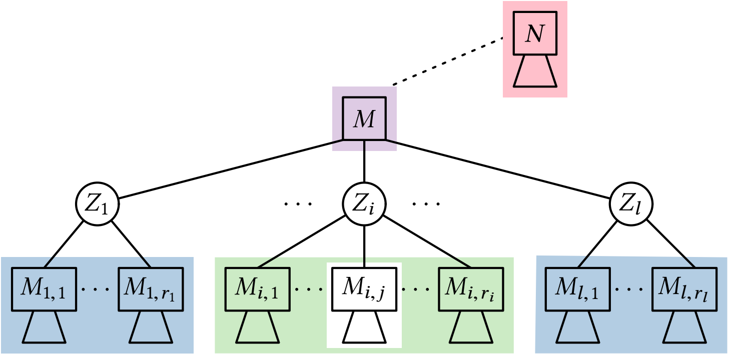


Figure 4.43: Aggregating information to support upward queries from a nested network $M_{i,j}$ in the s-t-component Z_i of a two-terminal network M with terminals s and t that is part of a larger two-terminal network N. We combine the data structures for inward downward queries (green) to the other nested networks of M in Z_i, outward downward queries (blue) to the nested networks of the other s-t-components of M, local queries (purple) to $M \setminus (Z_1 \cup \cdots \cup Z_l)$, and upward queries (red) to $N \setminus M$.

So far, we have described upward queries from the frame A_j of a nested network N_j of N that corresponds to a child node of the root in the nesting hierarchy $\mathcal{H}$ of N. In order to support upward queries from the descendants of N_j, we copy the data structure for queries in $N \setminus N_j$ along e_j to the nested networks in N_j. As illustrated in Figure 4.43, we support downward queries from a nested network N_{j_l} of N_j by combining the data structure for upwards queries from N_j via e_j with the data structure for the frame A_j of N_j, and the data structures for downward queries to the descendents of N_j that are reached from N_{j_l} by ascending through N_j.

Recall that τ is the number of nodes in the nesting hierarchy $\mathcal{H}$ of N. Every edge e of N appears a constant number of times in the data structures associated with each node of the nesting hierarchy. Therefore, the resulting data structure has the following properties.

Theorem 4.24. *Let N be a bi-connected two-terminal series-parallel network with n vertices, parallelism λ, nesting number τ, and nesting depth δ. There is a data structure with $O(\tau n + \delta n \log \lambda)$ construction time and $O(\tau n)$ size that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in N, where k is the number of reported farthest points.* □

4.6 Series-Parallel Networks

A series-parallel network is a network N where every bi-connected component B of N is a two-terminal series-parallel network, as illustrated in Figure 4.44.

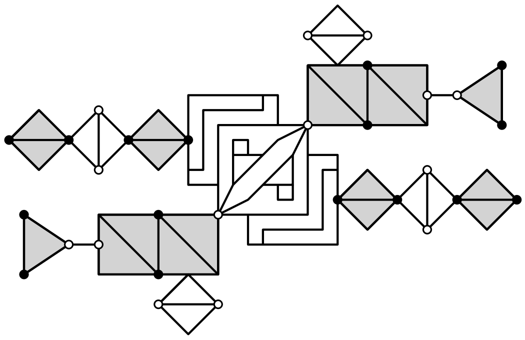
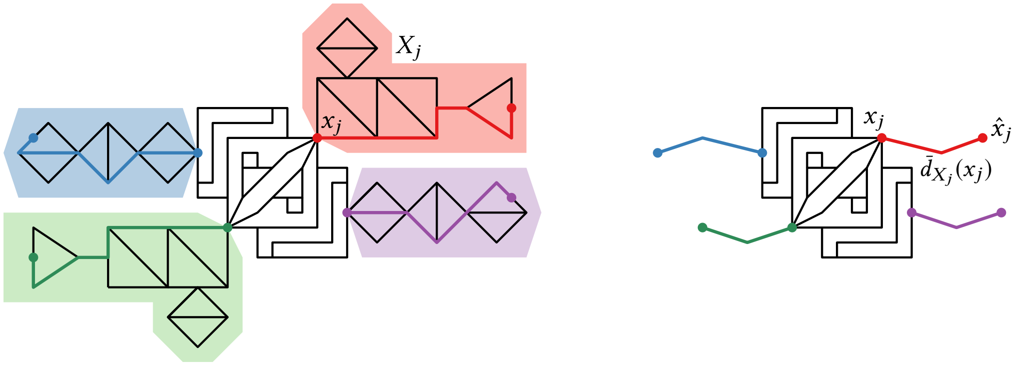


Figure 4.44: A series-parallel network with alternating white and grey bi-connected components. Each bi-connected component is two-terminal series-parallel and its terminals are indicated with solid discs (grey bi-connected components) or empty discs (white bi-connected components). Two bi-connected components are a single edge.

We consider a series-parallel network N with n vertices and b bi-connected components $B_1, B_2, \dots, B_b$. For $i = 1, 2, \dots, b$, let n_i be the number of vertices in B_i, let λ_i be the parallelism of B_i, and let τ_i be the nesting number of B_i. Let ζ be the number of cut vertices of N and let η be the number of incidences between bi-connected components. We have $\sum_{i=1}^{b} n_i = n + \eta - \zeta = O(n)$, since every cut vertex is counted once for each bi-connected component containing it. Let $\lambda = \max_{i=1}^{b} \lambda_i$ be the largest parallelism and let $\tau = \max_{i=1}^{b} \tau_i$ be the largest nesting number.



(a) The external components of B.

(b) The perspective from B on N.

Figure 4.45: A bi-connected component B in a series-parallel network N with four external components (coloured). In $\mathrm{per}_N(B)$, each external component X_j is replaced by an edge weighted with the farthest distance from the vertex $x_j = B \cap X_j$ into X_j.

For $i = 1, 2, \dots, b$, let $x_1, x_2, \dots, x_{\zeta_i}$ be the cut vertices of B_i, and let $X_1, X_2, \dots, X_{\zeta_i}$ be the connected components of $N \setminus B_i$ such that $x_j \in X_j$ for each $j = 1, 2, \dots, \zeta_i$. We call $X_1, X_2, \dots, X_{\zeta_i}$ the *external components* of B_i in N. The perspective $\mathrm{per}_N(B_i)$ from B_i onto N is the network that results from N by replacing X_j with an edge of weight $\bar{d}_{X_j}(x_j)$ from x_j to a new vertex $\hat{x}_j$, as illustrated in Figure 4.45. As shown in previous works [13, 27], the perspective from B_i onto N preserves the farthest-distance information from N for points on B_i, i.e., for all $q \in B_i$ we have $\bar{d}_N(q) = \bar{d}\mathrm{per}_N(B_i)(q)$. Furthermore, a point $q \in B_i$ has a farthest point in N that lies in X_j if and only if the point x'_j is farthest from q in $\mathrm{per}_N(B_i)$. Furthermore, if we have data structures that support $O(1)$-time farthest-distance queries and $O(k)$-time farthest point queries from the cut vertices of N in every bi-connected component and from the dummy vertices in the perspective of every bi-connected component, then we can combine these data structures to a data structure for queries in N with $O(n)$ additional time and space.

The perspectives from bi-connected components of a series-parallel network are two-terminal series-parallel networks with attached pendant edges. These networks are not two-terminal series-parallel themselves, so it would appear that we need an additional data structure. Fortunately, the data structure for two-terminal series-parallel networks already supports queries in these networks: If we double each of the pendant edges in the perspective $\mathrm{per}_N(B_i)$ of a bi-connected component B_i on N, then these two edges form an arc with identical endpoints, as illustrated in Figure 4.46. Notice that none of the constructions for bead-chains, abacus networks, or two-terminal series-parallel networks require the endpoints of the arcs to be distinct. This means we already have a data structure for queries in two-terminal series-parallel networks with attached pendant edges.

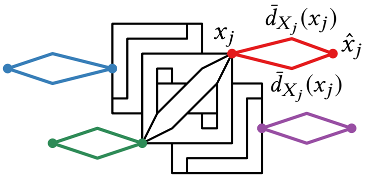


Figure 4.46: Replacing the external components of a bi-connected component B_i in N with loop arcs instead of pendant edges.

Corollary 4.25. *Let B_i be a two-terminal series-parallel network with n_i vertices, parallelism λ_i, and nesting number τ_i that is a bi-connected component in a series-parallel network N. Suppose we are given the perspective $\mathrm{per}_N(B_i)$ from B_i onto N. There is a data structure with $O(\tau_i n_i n_i \log \lambda_i)$ construction time and $O(\tau_i n_i)$ size that supports $O(\log n_i)$-time farthest-distance queries and $O(k + \log n_i)$-time farthest-point queries in $\mathrm{per}_N(B_i)$, where k is the number of reported farthest points.* □

We apply our approach for treelike networks to build a data structure for queries in N. First, we identify the bi-connected components and the cut vertices of N, which takes $O(n)$ time: The *bi-connected component-cutpoint-tree* T of N [37] is the tree whose vertices are the bi-connected components of N and the cut vertices of N; a bi-connected component B is connected to the cut vertex v in T when $v \in B$. Second, for each $i = 1, 2, \dots, b$, we identify the terminals of the bi-connected component B_i alongside with its nesting hierarchy using the algorithm from Theorem 4.19 and construct the data structure from Theorem 4.24 for queries in B_i. This step takes $O\left(\sum_{i=1}^{b} n_i \log \lambda_i\right) = O(n \log \lambda)$ total time. Third, we create the perspectives from each bi-connected component starting from some arbitrary bi-connected component B_1. Let

$X_1, X_2, \dots, X_{\zeta_1}$ be the external components of B_1 in N attached to the cut vertices $x_1, x_2, \dots, x_{\zeta_1}$ of N in B_1. If X_j consists of a single bi-connected component B_l, then we determine $\bar{d}_{X_j}(x_j) = \bar{d}_{B_l}(x_j)$ with a query from x_j in that bi-connected component. If X_j consists of ℓ bi-connected components $B_{l_1}, B_{l_2}, \dots, B_{l_\ell}$ that are all incident to B_i at x_j, then we have $\bar{d}_{X_j}(x_j) = \max_{r=1}^{\ell} \bar{d}_{B_{l_r}}(x_j)$. Otherwise, we recursively determine the farthest distance from x_j in X_j with a breadth-first search in the bi-connected component-cutpoint-tree T; this search returns when it reaches the leaves of T. We obtain the distances required to construct the perspective from B_1 onto N in $O\left(n + \sum_{i=2}^{b} \zeta_i \log n_i)\right)$ time, since the farthest-distance queries from the cut vertices in each bi-connected component B_i take $O(\log n_i)$-time and we perform $\zeta_i - 1$ queries in each B_i for $i = 2, 3, \dots, b$. Once we have the perspective from B_i onto N, we use it to obtain the missing distance required to construct the perspectives from the neighbours of B_i in the tree structure T. We propagate the distance information from T in a breadth-first-search fashion. This takes again $O\left(n + \sum_{i=1}^{b} \log n_i\right)$ time, since each bi-connected component B_i is only one missing the farthest distance into its external component that contains the starting bi-connected component B_1. This means that we can construct the perspectives from all bi-connected components in total time

$$O\left(n + \sum_{i=1}^{b} \zeta_i \log n_i + \sum_{i=1}^{b} \log n_i + \sum_{i=1}^{b} \tau_i n_i \log \lambda_i\right) = O(\tau n \log \lambda) \enspace .$$

With this preparation, we can answer a farthest-distance query from a point $q \in N$ in $O(\log n)$ time with a farthest-distance query from q in the perspective onto N from the bi-connected component B that contains q. For a farthest-point query from q, we first perform a farthest-point query from q in $\mathrm{per}_N(q)$. We output the reported farthest points from q in $\mathrm{per}_N(B)$ that lie on B and we cascade the query into the neighbouring bi-connected components of B as indicated by the reported dummy vertices. During the recursive construction of the perspectives from the bi-connected components, we introduce shortcuts in the tree structure that allow us to avoid cascading through long sequences of bi-connected components without any farthest points from q. For each edge in the tree structure, we store a shortcut to the next bi-connected component where we find a farthest point, or where two paths (in T) to bi-connected components containing farthest points from q split. We obtain these pointers as a by-product when calculating the distances into the external components for each bi-connected component [13, 27].

Theorem 4.26. *Let N be a series-parallel network with n vertices that consists of b bi-connected components with size $n_1, n_2, \dots, n_b$, nesting numbers $\tau_1, \tau_2, \dots, \tau_b$, and parallelisms $\lambda_1, \lambda_2, \dots, \lambda_b$, respectively, where $\tau = \max_{i=1}^{b} \tau_i$ and $\lambda = \max_{i=1}^{b} \lambda_i$ are the largest nesting number and parallelism. There exists a data structure with $O\left(\sum_{i=1}^{b} \tau_i n_i\right) = O(\tau n)$ size and $O\left(\sum_{i=1}^{b} \tau_i n_i \log \lambda_i\right) = O(\tau n \log \lambda)$ construction time that supports $O(\log n)$-time farthest-distance queries and $O(k + \log n)$-time farthest-point queries in N, where k is the number of reported farthest points.* ☐

5 Conclusion

We point out commonalities between the problems that we have studied independently thus far, we discuss ideas for improving some of our results, and other directions for future research.

5.1 Commonalities

We observed in Lemma 2.1 that adding a single shortcut pq to a geometric cycle C creates two regions along C that are unaffected by the shortcut. As illustrated in Figure 5.1, these regions are the mirror image of the shortcut along the two cycles in $C + pq$ that contain pq. Thus, these regions have the same length as the shortcut pq. This observation is, in fact, a special case of the observation from Lemmas 4.1 and 4.2 where we characterize which terminals a query point on a parallel-path network is facing: the unaffected regions correspond to queries that are facing both terminals. The description of the unaffected regions in $C + pq$ follows from the characterization of the query points that are facing both ways when interpreting $C + pq$ as a parallel-path network with parallelism $\lambda = 3$ where the shortest u-v-path is the straight-line segment pq.

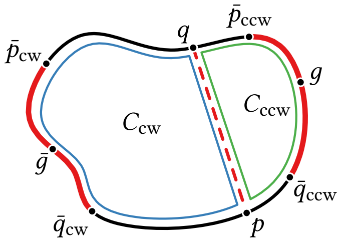


Figure 5.1: The unaffected regions along a geometric cycle C when augmenting C with a shortcut pq.

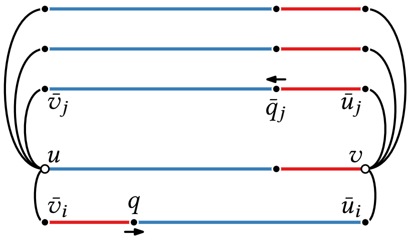
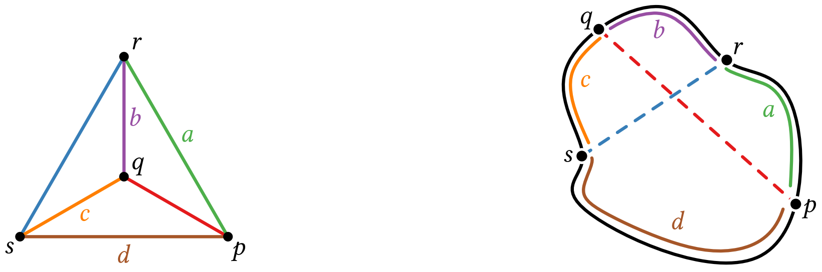


Figure 5.2: The region of query points along a parallel-path network that are facing both terminals.

Although the details differ substantially, the algorithm for placing an optimal pair of shortcuts for a cycle from Chapter 2 and the algorithm for placing an optimal shortcut for a tree from Chapter 3 follow the same strategy. First, we restrict the search space, viz., to shortcuts in alternating configuration for cycles and to shortcuts with both endpoints along the backbone that span across the absolute center for trees. Second, we characterize configurations of diametral paths in the augmented network that define algorithm states and we show that optimal shortcuts balance diametral paths. Third, we derive rules that inform us how to slide the shortcuts along the search space in a way that we encounter optimal shortcuts. Fourth, we discretize the continuous algorithm by identifying events where the algorithm states change. Fifth, we bound the running time by bounding the number of events and the time required to handle each event.



(a) The complete network with four vertices K_4.

(b) Two alternating shortcuts for a cycle.

Figure 5.3: The correspondence between an augmented cycle $C + pq + rs$ with two shortcuts in alternating configuration and a complete network K_4 with four vertices.

We observed that we require two shortcuts to reduce the continuous diameter of a cycle and that an optimal pair of shortcuts is alternating. A cycle with two shortcuts in alternating configuration is a subdivision of K_4, i.e., the complete graph with four vertices, as illustrated in Figure 5.3. However, K_4 is the forbidden minor for series-parallel networks. This may pose a challenge when we wish to create a dynamic data structure for farthest-point queries or when we wish to use a data structure for farthest-point queries as an aid to place multiple shortcuts.

5.2 Potential Improvements

The algorithm from Chapter 2 takes $O(k^2 n)$ time to determine an optimal pair of shortcuts for a non-convex cycle with n vertices and k reflex vertices. This running time is dominated by the search for an optimal pair of shortcuts where both shortcuts have one endpoint at a reflex vertex. For each of the $\binom{k}{2}$ pairs of reflex vertices, we scan a path of length $O(n)$ for an optimal position of the other endpoints of the shortcuts. We conjecture that we can combine these searches into one search that takes $O(kn \log n)$ time, where we slide a candidate for the midpoint of the longest section along the cycle and maintain for each reflex vertex the position (or positions) of an optimal target for a shortcut from that reflex vertex, as illustrated in Figure 5.4.

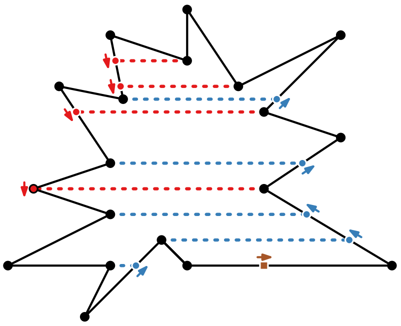
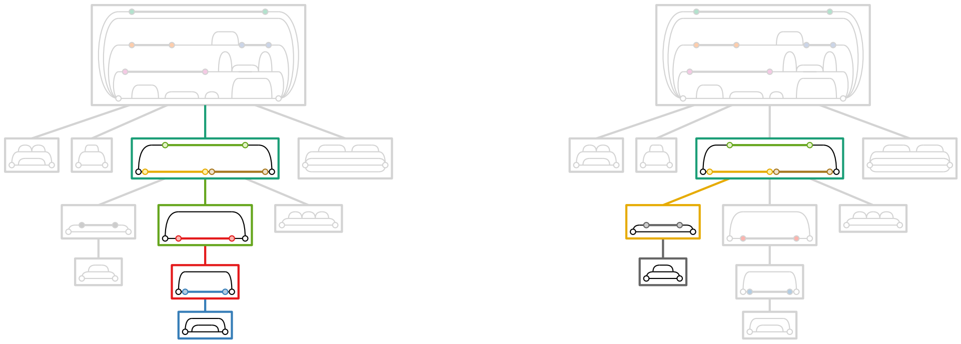


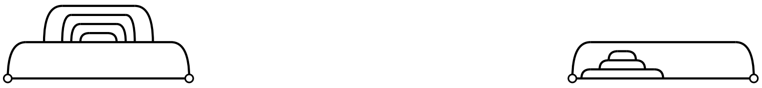
Figure 5.4: A sketch of a simultaneous search for the best pair of shortcuts for a non-convex polygonal cycle where both shortcuts have an endpoint at a reflex vertex.

We conjecture that we can reduce the dependence on the size τ of the nesting hierarchy in the construction time of a data structure for farthest-point queries in a two-terminal series-parallel network by applying path compression to the nesting hierarchy. A compressible path in the nesting hierarchy corresponds to one of the two networks depicted in Figure 5.5; both are similar to parallel-path networks. The challenge would be to devise a data structure that reports which neighbouring paths in the nesting hierarchy contain farthest points from a query.



(a) A path $\mathcal{P}_1$ in the nesting hierarhcy.

(b) A path $\mathcal{P}_2$ in the nesting hierarchy.



(c) The network corresponding to $\mathcal{P}_1$.

(d) The network corresponding to $\mathcal{P}_2$.

Figure 5.5: Two paths in the nesting hierarchy of a two-terminal series-parallel network.

5.3 Future Research

Future research on farthest-distance queries in networks would be concerned with more general types of networks. We could investigate dynamic data structures, where we add or remove edges or modify edge weights or approximate farthest-distance queries. Instead of querying for the farthest points from q, we could query for the points at distance θ from q for some value θ.

There are numerous directions for future research on minimum-diameter network augmentation. We could minimize the continuous diameter when augmenting a cycle with $k \geq 3$ shortcuts or a tree with $k \geq 2$ shortcuts. We could investigate which cycles and trees benefit from k shortcuts and characterize optimal configurations of shortcuts. We could study planar variants, where shortcuts must not cross the network or where every crossing becomes a vertex. We could add shortcuts one at a time and allow the endpoints of a shortcut to connect to points on a previous shortcut. At WADS 2017, Therese Biedl suggested minimizing the continuous diameter of a geometric tree with edge weights. At CCCG 2017, Joseph O'Rourke proposed a three-dimensional version of the augmentation problem for cycles where we minimize the geodesic diameter when augmenting a sphere with chords.

Bibliography

[1] Manuel Abellanas, Ferran Hurtado, Vera Sacristan, Christian Icking, Lihong Ma, Rolf Klein, Elmar Langetepe, and Belén Palop. “Voronoi Diagram for services neighboring a highway”. In: *Information Processing Letters* 86.5 (2003), pp. 283–288. DOI: `10.1016/S0020-0190(02)00505-7`.

[2] Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter Shor, and Robert Wilber. “Geometric Applications of a Matrix-Searching Algorithm”. In: *Algorithmica* 2.1 (1987), pp. 195–208. DOI: `10.1007/BF01840359`.

[3] Oswin Aichholzer, Franz Aurenhammer, and Belén Palop. “Quickest Paths, Straight Skeletons, and the City Voronoi Diagram”. In: *Discrete & Computational Geometry* 31.1 (2004), pp. 17–35. DOI: `10.1007/s00454-003-2947-0`.

[4] Stefan Arnborg, Jens Lagergren, and Detlef Seese. “Easy problems for tree-decomposable graphs”. In: *Journal of Algorithms* 12.2 (1991), pp. 308–340. DOI: `10.1016/0196-6774(91)90006-K`.

[5] Boris Aronov, Kevin Buchin, Maike Buchin, Bart Jansen, Tom de Jong, Marc van Kreveld, Maarten Löffler, Jun Luo, Rodrigo Silveira, and Bettina Speckmann. “Connect the dot: Computing feed-links for network extension”. In: *Journal of Spatial Information Science* 3.1 (2011), pp. 3–31. DOI: `10.5311/JOSIS.2011.3.47`.

[6] Sang Won Bae and Kyung-Yong Chwa. “Farthest Voronoi Diagrams under Travel Time Metrics”. In: *Proceedings of the 6th International Workshop on Algorithms and Computation.* WALCOM 2012. (Dhaka, Bangladesh, Feb. 15–17, 2012). 2012, pp. 28–39. DOI: `10.1007/978-3-642-28076-4_6`.

[7] Sang Won Bae and Kyung-Yong Chwa. “Shortest Paths and Voronoi Diagrams with Transportation Networks Under General Distances”. In: *Proceedings of the 16th International Symposium on Algorithms and Computation.* ISAAC 2005. (Sanya, Hainan, China, Dec. 19–21, 2005). 2005, pp. 1007–1018. DOI: `10.1007/11602613_100`.

[8] Sang Won Bae and Kyung-Yong Chwa. “Voronoi Diagrams for a Transportation Network on the Euclidean Plane”. In: *International Journal of Computational Geometry & Applications* 16.2-3 (2006), pp. 117–144. DOI: `10.1142/S0218195906001963`.

[9] Boaz Ben-Moshe, Binay Bhattacharya, Qiaosheng Shi, and Arie Tamir. “Efficient algorithms for center problems in cactus networks”. In: *Theoretical Computer Science* 378.3 (2007), pp. 237–252. DOI: `10.1016/j.tcs.2007.02.033`.

[10] Marshall W. Bern, Eugene L. Lawler, and A. L. Wong. “Linear-Time Computation of Optimal Subgraphs of Decomposable Graphs”. In: *Journal of Algorithms* 8.2 (1987), pp. 216–235. DOI: `10.1016/0196-6774(87)90039-3`.

[11] Hans L. Bodlaender. "A Partial k-Arboretum of Graphs with Bounded Treewidth". In: *Theoretical Computer Science* 209.1-2 (1998), pp. 1–45. DOI: `10.1016/S0304-3975(97)00228-4`.

[12] Prosenjit Bose, Kai Dannies, Jean-Lou De Carufel, Christoph Doell, Carsten Grimm, Anil Maheshwari, Stefan Schirra, and Michiel Smid. "Network Farthest-Point Diagrams". In: *Journal of Computational Geometry* 4.1 (2013), pp. 182–211. DOI: `10.20382/jocg.v4i1a8`.

[13] Prosenjit Bose, Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, and Michiel Smid. "Optimal Data Structures for Farthest-Point Queries in Cactus Networks". In: *Journal of Graph Algorithms and Applications* 19.1 (2015), pp. 11–41. DOI: `10.7155/jgaa.00345`.

[14] José Cáceres, Delia Garijo, Antonio González, Alberto Márquez, María Luz Puertas, and P. Ribeiro. "Shortcut Sets for Plane Euclidean Networks". In: *Electronic Notes in Discrete Mathematics* 54 (2016), pp. 163–168. DOI: `10.1016/j.endm.2016.09.029`.

[15] Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, and Michiel Smid. "Minimizing the Continuous Diameter when Augmenting Paths and Cycles with Shortcuts". In: *Proceedings of the 15th Scandinavian Symposium and Workshops on Algorithm Theory*. SWAT 2016. (Reykjavik, Iceland, June 22–24, 2016). 2016, 27:1–27:14. DOI: `10.4230/LIPIcs.SWAT.2016.27`.

[16] Jean-Lou De Carufel, Carsten Grimm, Stefan Schirra, and Michiel Smid. "Minimizing the Continuous Diameter When Augmenting a Tree with a Shortcut". In: *Proceedigns of the 15th International Symposium on Algorithms and Data Structures*. WADS 2017. (St. John's, Newfoundland and Labrador, Canada, July 31–Aug. 2, 2017). 2017, pp. 301–312. DOI: `10.1007/978-3-319-62127-2_26`.

[17] Bernard Chazelle. "Filtering Search: A New Approach to Query-Answering". In: *SIAM Journal on Computing* 15.3 (1986), pp. 703–724. DOI: `10.1137/0215051`.

[18] Bernard Chazelle and Leonidas J. Guibas. "Fractional Cascading: I. A Data Structuring Technique". In: *Algorithmica* 1.2 (1986), pp. 133–162. DOI: `10.1007/BF01840440`.

[19] Victor Chepoi and Yann Vaxès. "Augmenting Trees to Meet Biconnectivity and Diameter Constraints". In: *Algorithmica* 33.2 (2002), pp. 243–262. DOI: `10.1007/s00453-001-0113-8`.

[20] Richard J. Duffin. "Topology of Series-Parallel Networks". In: *Journal of Mathematical Analysis and Applications* 10.2 (1965), pp. 303–318. DOI: `10.1016/0022-247X(65)90125-3`.

[21] Martin Erwig. "The graph Voronoi diagram with applications". In: *Networks* 36.3 (2000), pp. 156–163. DOI: `10.1002/1097-0037(200010)36:3⟨156::AID-NET2⟩3.0.CO;2-L`.

[22] Chenglin Fan, Jun Luo, and Binhai Zhu. "Continuous-Time Moving Network Voronoi Diagram". In: *Transactions on Computational Science XIV. Special Issue on Voronoi Diagrams and Delaunay Triangulation*. Vol. 6970. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 129–150. DOI: `10.1007/978-3-642-25249-5_5`.

[23] Mohammad Farshi, Panos Giannopoulos, and Joachim Gudmundsson. "Improving the Stretch Factor of a Geometric Network by Edge Augmentation". In: *SIAM Journal on Computing* 38.1 (2008), pp. 226–240. DOI: `10.1137/050635675`.

[24] Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. "Augmenting Graphs to Minimize the Diameter". In: *Algorithmica* 72.4 (2015), pp. 995–1010. DOI: `10.1007/s00453-014-9886-4`.

[25] Greg N. Frederickson. "Parametric search and locating supply centers in trees". In: *Proceedings of the 2nd Workshop on Algorithms and Datastructures.* WADS '91. (Ottawa, Canada, Aug. 14–16, 1991). 1991, pp. 299–319. DOI: `10.1007/BFb0028271`.

[26] Michael L. Fredman and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *Journal of the ACM* 34.3 (1987), pp. 596–615. DOI: `10.1145/28869.28874`.

[27] Carsten Grimm. "Eccentricity Diagrams. On Charting Farthest-Point Information in Networks". Master's Thesis (Diplomarbeit). Magdeburg, Germany: Otto-von-Guericke-Universität Magdeburg, Mar. 2012.

[28] Carsten Grimm. "Efficient Farthest-Point Queries in Two-Terminal Series-Parallel Networks". In: *Proceedings of the 41st International Workshop on Graph-Theoretic Concepts in Computer Science.* WG 2015. (Garching bei München, Germany, July 17–19, 2015). 2015, pp. 122–137. DOI: `10.1007/978-3-662-53174-7_10`.

[29] Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel Smid, and Fabian Stehn. "Fast Algorithms for Diameter-Optimally Augmenting Paths". In: *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming.* ICALP 2015. (Kyoto, Japan, July 6–10, 2015). 2015, pp. 678–688. DOI: `10.1007/978-3-662-47672-7_55`.

[30] Ulrike Große, Joachim Gudmundsson, Christian Knauer, Michiel Smid, and Fabian Stehn. *Fast Algorithms for Diameter-Optimally Augmenting Paths and Trees.* 2016. arXiv: `1607.05547 [cs.CG]`.

[31] Yuri Gurevich, Larry Stockmeyer, and Uzi Vishkin. "Solving NP-hard problems on graphs that are almost trees and an application to facility location problems". In: *Journal of the Association for Computing Machinery* 31.3 (1984), pp. 459–473. DOI: `10.1145/828.322439`.

[32] S. Louis Hakimi. "Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph". In: *Operations Research* 12.3 (1964), pp. 450–459. JSTOR: `168125`.

[33] S. Louis Hakimi, Martine Labbé, and Edward Schmeichel. "The Voronoi partition of a network and its implications in location theory". In: *ORSA Journal on Computing* 4.4 (1992), pp. 412–417. DOI: `10.1287/ijoc.4.4.412`.

[34] Jonathan Halpern and Oded Maimon. "Algorithms for the m-center problems: A survey". In: *European Journal of Operational Research* 10.1 (1982), pp. 90–99. DOI: `10.1016/0377-2217(82)90136-9`.

[35] Pentti Hämäläinen. "The absolute center of a unicyclic network". In: *Discrete Applied Mathematics* 25.3 (1989), pp. 311–315. DOI: `10.1016/0166-218X(89)90009-7`.

[36] Pierre Hansen, Martine Labbé, and Brigitte Nicolas. “The Continuous Center Set of a Network”. In: *Discrete Applied Mathematics* 30.2-3 (1991), pp. 181–195. DOI: `10.1016/0166-218X(91)90043-V`.

[37] Frank Harary and Geert Prins. “The block-cutpoint-tree of a graph”. In: *Publicationes Mathematicae Debrecen* 13 (1966), pp. 103–107.

[38] John Hopcroft and Robert Tarjan. “Algorithm 447: efficient algorithms for graph manipulation”. In: *Communications of the ACM* 16 (6 1973), pp. 372–378. DOI: `10.1145/362248.362272`.

[39] Ferran Hurtado, Rolf Klein, Elmar Langetepe, and Vera Sacristán. “The weighted farthest color Voronoi diagram on trees and graphs”. In: *Computational Geometry* 27.1 (2004), pp. 13–26. DOI: `10.1016/j.comgeo.2003.07.003`.

[40] Oded Kariv and S. Louis Hakimi. “An Algorithmic Approach to Network Location Problems. I: The p-Centers”. In: *SIAM Journal on Applied Mathematics* 37.3 (1979), pp. 513–538. DOI: `10.1137/0137040`.

[41] Rex K. Kincaid. “Exploiting Structure: Location Problems on Trees and Treelike Graphs”. In: *Foundations of Location Analysis*. 2011, pp. 315–334. DOI: `10.1007/978-1-4419-7572-0_14`.

[42] Yu-Feng Lan, Yue-Li Wang, and Hitoshi Suzuki. “A linear-time algorithm for solving the center problem on weighted cactus graphs”. In: *Information Processing Letters* 71.5–6 (1999), pp. 205–212. DOI: `10.1016/S0020-0190(99)00111-8`.

[43] 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”. In: *Operations Research Letters* 11.5 (1992), pp. 303–308. DOI: `10.1016/0167-6377(92)90007-P`.

[44] Jun Luo and Christian Wulff-Nilsen. “Computing Best and Worst Shortcuts of Graphs Embedded in Metric Spaces”. In: *19th International Symposium on Algorithms and Computation*. ISAAC 2008. (Gold Coast, Australia, Dec. 15–17, 2008). 2008, pp. 764–775. DOI: `10.1007/978-3-540-92182-0_67`.

[45] Nimrod Megiddo and Arie Tamir. “New Results on the Complexity of p-Center Problems”. In: *SIAM Journal on Computing* 12.4 (1983), pp. 751–758. DOI: `10.1137/0212051`.

[46] Eunjin Oh and Hee-Kap 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 2016. (Sydney, Australia, Dec. 12–14, 2016). 2016, 59:1–59:12. DOI: `10.4230/LIPIcs.ISAAC.2016.59`.

[47] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*. 2nd ed. John Wiley & Sons Ltd., 2000.

[48] Atsuyuki Okabe, Toshiaki Satoh, Takehiro Furuta, Atsuo Suzuki, and K. Okano. "Generalized Network Voronoi Diagrams: Concepts, Computational Methods, and Applications". In: *International Journal of Geographical Information Science* 22.9 (2008), pp. 965–994. DOI: `10.1080/13658810701587891`.

[49] Charles S. ReVelle and Horst A. Eiselt. "Location analysis: A synthesis and survey". In: *European Journal of Operational Research* 165.1 (2005), pp. 1–19. DOI: `10.1016/j.ejor.2003.11.032`.

[50] H. N. de Ridder et al. *Graphclass: 2-terminal Series-Parallel.* Information System on Graph Classes and their Inclusions (ISGCI). July 7, 2016. URL: `http://www.graphclasses.org/classes/gc_875.html` (visited on 07/03/2017).

[51] H. N. de Ridder et al. *Graphclass: Series-Parallel.* Information System on Graph Classes and their Inclusions (ISGCI). July 7, 2016. URL: `http://www.graphclasses.org/classes/gc_275.html` (visited on 07/03/2017).

[52] Anneke A. Schoone, Hans L. Bodlaender, and Jan van Leeuwen. "Diameter Increase Caused by Edge Deletion". In: *Journal of Graph Theory* 11.3 (1987), pp. 409–427. DOI: `10.1002/jgt.3190110315`.

[53] Qiaosheng Shi. "Efficient algorithms for network center/covering location optimization problems". PhD thesis. School of Computing Science-Simon Fraser University, 2008.

[54] Qiaosheng Shi and Binay K. Bhattacharya. "Application of computational geometry to network p-center location problems". In: *Proceedings of the 20th Canadian Conference on Computational Geometry.* CCCG 2008. (Montréal, Canada, Aug. 13–15, 2008).

[55] K. Takamizawa, Takao Nishizeki, and Nobuji Saito. "Linear-time computability of combinatorial problems on series-parallel graphs". In: *Journal of the ACM* 29.3 (1982), pp. 623–641. DOI: `10.1145/322326.322328`.

[56] Barbaros Ç. Tansel. "Discrete Center Problems". In: *Foundations of Location Analysis.* 2011, pp. 79–106. DOI: `10.1007/978-1-4419-7572-0_5`.

[57] Barbaros Ç. Tansel, Richard L. Francis, and Timothy J. Lowe. "Location on Networks: A Survey. Part I: The p-Center and p-Median Problems". In: *Management Science* 29.4 (1983), pp. 482–497. JSTOR: `2630870`.

[58] Barbaros Ç. Tansel, Richard L. Francis, and Timothy J. Lowe. "Location on Networks: A Survey. Part II: Exploiting Tree Network Structure". In: *Management Science* 29.4 (1983), pp. 498–511. JSTOR: `2630871`.

[59] Joseph A. Wald and Charles J. Colbourn. "Steiner trees, partial 2-trees, and minimum IFI networks". In: *Networks* 13.2 (1983), pp. 159–167. DOI: `10.1002/net.3230130202`.

[60] Haitao Wang. "An Improved Algorithm for Diameter-Optimally Augmenting Paths in a Metric Space". In: *Proceedings of the 15th International Symposium on Algorithms and Data Structures.* WADS 2017. (St. John's, Newfoundland and Labrador, Canada, July 31–Aug. 2, 2017). 2017, pp. 545–556. DOI: `10.1007/978-3-319-62127-2_46`.

[61] Boting Yang. “Euclidean Chains and their Shortcuts”. In: *Theoretical Computer Science* 497 (2013), pp. 55–67. DOI: `10.1016/j.tcs.2012.03.021`.

[62] Jianzhong Zhang, Xiaoguang Yang, and Mao-cheng Cai. “Reverse Center Location Problem”. In: *Proceedings of the 10th International Symposium on Algorithms and Computation.* ISAAC’99. (Chennai, India, Dec. 16–18, 1999). 1999, pp. 279–294. DOI: `10.1007/3-540-46632-0_29`.

Ehrenerklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, den 1. Februar 2018

Carsten Grimm